9. Mr. Jeff Rish, Code R23 1
NSWCDD CSS
6703 West Highway 98
Panama City, FL 32407-7001

10. Dr. T. Swean 1
Office of Naval Research
800 N. Quincy St.
Arlington, VA 22217-5660

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center...2
   8725 John J. Kingman Road, Suite 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, CA 93943-5101

3. Professor Anthony J. Healey, Code ME/He...2
   Naval Postgraduate School
   700 Dyer Road
   Monterey, California 93943-5101

4. Mechanical Engineering Department Chairman, Code ME...................................1
   Naval Postgraduate School
   700 Dyer Road
   Monterey, California 93943-5101

5. Engineering & Technology Curricular Office, Code 34..1
   Naval Postgraduate School
   700 Dyer Road
   Monterey, California 93943-5101

6. LT Peter M. Ludwig..1
   674 Paden Drive
   Birmingham, Alabama 35226

7. LT Peter M. Ludwig..2
   SRF DET SASEBO
   PSC 476 Box 16
   FPO AP 96322-1400

8. Mr. Leolan H. Fry, Code R23 1
   NSWCDD CSS
   6703 West Highway 98
   Panama City, FL 32407-7001

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Crute, Daniel A., "Naval Mine Warfare Vision 2010: A View Toward the Future" [http://www.ncsc.navy.mil/CSS/Papers/vison.htm]. May 2000.

Fry, Leolan H. Jr., Coastal Systems Station, Panama City, FL, e-mail letter to Anthony J. Healey , Subject: Comments to swimmer sweep draft report, 31 Jan 2000.

Healey, A.J., Kim, J., "Multiple Autonomous Vehicle Solutions to Minefield Reconnaissance and Mapping" Proceedings of Australian-American Joint Conference on the Technologies of Mines and Mine Countermeasures, Sydney, Australia, July 12-16, 1999.

Kim, Joung K., "Magsweep_rpt," draft report presented to Professor Anthony J. Healey, Naval Postgraduate School, Monterey, California, January 2000.

Koopman, Bernard O., *Search and Screening: general principles with historical applications*, Pergamon Press, 1980.

Stone, J.A., *Theory of Optimal Search*, Academic Press, 1975.

```
The average searched area in iteration 100 is:          98.53%
The number of swimmers destroyed in iteration 100 is:     11
   11 targets of    11 cleared in    7866 steps

The average percent of cleared targets over all iterations is:     99.545%
The average searched space over all iterations is:    98.86%
```

Program Output:

In the process of running the simulation, the program generates several output files to which it writes data. The number of vehicles initially assigned to the simulation will replace any xxx seen in the file names. These files are used for data display and analysis. The most significant of these files are:

x.out contains one column for every vehicle and records the vehicle's x geographical position.

y.out contains one column for every vehicle and records the vehicle's y geographical position.

pvacrxxx.rst, which contains seven columns and one, row for every iteration, plus a final row for the average values. The columns are allocated as shown in the below figure.

Iteration number	Number of mines in the run	Number of mines cleared	Percent of mines cleared	Percent area searched	Number of swimmer vehicles used	Number of dead swimmers

Figure C.1 Columns for output file pvacrxxx.rst

fldvaxxx.rst contains the data needed to graphically display the areas that were left un-searched during the simulation (holidays).

target.out holds the data on the mines, including their geographical positions.

```
The vehicle characteristics for this simulation are:
    Speed is:              2.572222 m/sec
    Sensor range is:    4.267200 m
    GPS error is:         2.000000 m
    Charge time is:     15 sec

The spacing between vehicles is 2.133600 m.
    percent overlap is:  100.00%

Is this the desired overlap? (y or n)
n

Enter the desired vehicle interval: (in meters)
2.5

The spacing between vehicles is 2.500000 m.
    percent overlap is:   70.69%

Is this the desired overlap? (y or n)
y

The delay time between vehicles starting is:  35 sec

Input the number of seconds you want to run the simulation for.
The maximum time the program will run is 21600 seconds.
An input value of zero will default the simulation to maximum.
7200
```

The subsequent output at the conclusion of the simulation would appear like:

```
vehs = 19
vehs id = 0
fleet   = 1
vehs[i].pnew[X] = 53.500000
vehs[i].pnew[Y] = 0.000000
vehs[i].dgps[X] = 52.937389
vehs[i].dgps[Y] = 1.117366
vehs[i].speed   = 2.572222
vehs[i].hm_fl   = 0
vehs[i].clock   = -350
vehs[i].band_wd = 25.000000
```

After each iteration is completed, the program displays the iteration statistics as follows:

```
The average searched area in iteration <iteration #> is:<% searched>%
The number of swimmers destroyed in iteration <iteration #> is:<# killed>
<mines destroyed> targets of <total  mines> cleared in <#  steps> steps
```

Subsequently, after all iterations are run, the program averages all results and displays:

```
The average percent of cleared targets over all iterations is:<%cleared>%
The average searched space over all iterations is:<avrg % searched>%
```

<u>Example:</u>

Table A.1 below contains sample parameters for a simulation.

# of Vehicles	# of Iterations	Vehicle Speed	Sensor Width	Vehicle Spacing	% Overlap	Simulation Time
20	100	5 KTS	4.2672 m (14 ft)	2.5 m (8.202 ft)	70.69 %	7200 sec

Table A.1 Example Simulation Parameters

Using the above-tabulated parameters, the entire sequence for program initialization would appear as follows :

```
fry31_final_rev 20 100

The number of vehicles for this simulation is:   20
The number of iterations for this simulation is: 100

Input the desired vehicle speed: (in knots)
5

Input the desired sensor range of the vehicle: (in meters)
4.2672

The simulation field is:
    150.88 m by  2764.84 m
```

Next, the user will be prompted to verify if the default values for vehicle spacing and overlap as just shown are the desired values:

```
Is this the desired overlap? (y or n)
<y or n>
```

An answer of no will initiate the following additional questions:

```
Enter the desired vehicle interval: (in meters)
<vehicle spacing>

The spacing between vehicles is <vehicle spacing> m.
percent overlap is:   <calculated value>%

Is this the desired overlap? (y or n)
<y or n>
```

At this point, an additional answer of no will re-initiate the above sequence until an answer of yes is received. When the user accepts the overlap value, the program displays the delay time between vehicle starts and prompts for the simulation run time in the following sequence:

```
The delay time between vehicles starting is:  <calculated> sec

Input the number of seconds you want to run the simulation for.
The maximum time the program will run is 21600 seconds.
An input value of zero will default the simulation to maximum.
<simulation time>
```

As soon as the program receives the time input, it commences the simulation. As in the below example, for each vehicle in every iteration, the following information is displayed:

```
vehs = 19
vehs id = 0
fleet   = 1
vehs[i].pnew[X] = 53.500000
vehs[i].pnew[Y] = 0.000000
vehs[i].dgps[X] = 54.283245
vehs[i].dgps[Y] = -0.557471
vehs[i].speed   = 3.601111
vehs[i].hm_fl   = 0
vehs[i].clock   = -290
vehs[i].band_wd = 25.000000
```

Users Guide

Note:

For the purposes of this guide, the program name fry31_final_rev is used when required. Any name assigned during the compilation process may be substituted.

Running the Program:

The program is designed to be fairly user friendly, provided the user has knowledge of the subject matter and is familiar with the terminology used. To initially start running the program, at the command prompt, type:

```
fry31_final_rev <# of Vehicles> <# of Iterations>
```

So that the arguments may be verified, the program then prints the following verification statements to the screen and additionally prompts for the desired vehicle speed of the simulation:

```
The number of vehicles for this simulation is: <# of Vehicles>
The number of iterations for this simulation is: <# of Iterations>

Input the desired vehicle speed: (in knots)
<vehicle speed>
```

The program then prompts for the desired vehicle swath width to be input:

```
Input the desired sensor range of the vehicle: (in meters)
<sensor width>
```

After receiving the required input, the following data is printed to the screen:

```
The simulation field is:
150.88 m by  2764.84 m

The vehicle characteristics for this simulation are:
Speed is:           <vehicle speed> m/sec
Sensor range is:  <sensor width> m
GPS error is:        2.000000 m
Charge time is:    15 sec

The spacing between vehicles is <1/2 sensor width> m.
percent overlap is:  100.00%
```

APPENDIX C. USERS GUIDE TO THE PROGRAM

This appendix is intended to be utilized as a users guide to the compiled program. The basis for this guide was taken from a similar guide by Kim (2000) and included in the draft report generated after initially programming the file. It is intended to help step a user through the actual workings of the program and be a supplement to the material contained in the actual thesis, as a fundamental knowledge of the programs backbone is required in order to obtain desired results.

reallocation.m

```
% This m-file is specific for use with the x.out and y.out files contained in the folder
% labeled new_dx.  It is used to plot the paths of two specific vehicles chosen to show
% track re-allocation techniques.  It can be modified for use with other files by altering the
% numbers of the vehicles chosen and adjusting the axis sizes to accommodate the desired
% viewing range.


load x.out;
load y.out;
[rows,columns]=size(x);

figure
hold on
plot(x(:,9),y(:,9),'r.-')
plot(x(:,10),y(:,10),'b.-')
axis([70 150 -250 3500])
grid
t=['PATHS FOR VEHICLES #9 & 10'];
title(t)
legend('vehicle 9','vehicle 10')

figure
hold on
plot(x(:,9),y(:,9),'r.-')
plot(x(:,10),y(:,10),'b.-')
axis([70 115 -100 3000])
grid
title(t)
legend('vehicle 9','vehicle 10')
```

graphs.m

```
% This m-file is a generic file used to plot the output data from the files x.out and y.out
% generated by the program fry31_rev_final.c.  It is designed to plot the results for any
% number of vehicles chosen for simulation.


load x.out;
load y.out;
[rows,columns]=size(x);

plot(x,y)
axis([-5 170 -250 3000])
grid
t1=['ALL VEHICLE PATHS DISPLAYED, ', num2str(columns), 'VEHICLES'];
title(t1)

for n=1:columns
   figure
   plot(x(:,n),y(:,n),'r.')
   axis([-5 170 -250 3000])
   grid
   t2=['PATH FOR VEHICLE #', num2str(n),' turn time=1'];
   title(t2)
end
```

APPENDIX B. *MATLAB* FILES FOR GRAPHICAL DATA DISPLAY

This appendix contains the *Matlab* files utilized to generate graphical displays of the output data from the files x.out and y.out generated during the simulations.

```
                tgt[i].atr    = 1;
                tgt[i].sch_flg = 0;
                Line[X]  += 50.0;
                i++;
            }

            /* Place targets on the Mine field # 2 */

            Line[Y] = max_Y_length - 2304 * FT;
            Line[X] = 20.0 * (rderr() + 3.0) / 6.0;

            while (Line[X] <= max_X_length)
          {
             tgt[i].pos[X] = Line[X] + pos_err();
             tgt[i].pos[Y] = Line[Y] + pos_err();
                tgt[i].atr    = 2;
                tgt[i].sch_flg = 0;
             Line[X]  += 20.0;
             i++;
          }

          no_target = i;

          inptr = fopen("target.out","w");
          for (i=0; i<no_target; i++) {
                 fprintf(inptr, "%10.5f %10.5f ", tgt[i].pos[X], tgt[i].pos[Y]);
                       fprintf(inptr, "%3d %3d \n", tgt[i].atr, tgt[i].sch_flg);
            }
          fclose(inptr);
    }
```

```
                    vehs[v].dest[X] = Dx;
                }
              }
                    vehs[v].gwp_flg = 0;
                    break;
          }
}

float pos_err( )
{
/*        Compute the Target position error     */

          return(rderr() * tg_pos_err / 3.0);

}

void init_target()
/*-------------------------------------------------------------------- */
/*    Function:  init_target                                        */
/*    Parameters:                                                  */
/*    set the initial target position                                 */
/*-------------------------------------------------------------------- */
{
        int i;
          float FT, Line[XY];

        FILE *inptr;
/*                              */
/*  Initialize the targets position */
/*                              */

          FT = 0.3048;

          /* Place targets on the minefield #1 */

          Line[Y] = max_Y_length - 4956.0 * FT;
          Line[X] = 50.0 * (rderr() + 3.0) / 6.0;

          i = 0;

          while (Line[X] <= max_X_length)
          {
             tgt[i].pos[X] = Line[X] + pos_err();
             tgt[i].pos[Y] = Line[Y] + pos_err();
```

```
                    vehs[v].dest[X] = max_X_length - dif;
                    vehs[v].flag = 1;
                  }
                  else {
              vehs[v].dest[X] = Dx;
                  }
          }
            }
            vehs[v].gwp_flg = 0;
break;
case 2:
            vehs[v].gwp_flg = 1;
            vehs[v].dest[Y] = 0.0;
            break;
   default:
            if (vehs[v].flag) {
              Dx = vehs[v].id * vh_int;
          if (zn) {
             vehs[v].dest[X] += Dx;
             vehs[v].dir = 0;
          }
          else {
             vehs[v].dest[X] -= Dx;
             vehs[v].dir = 1;
          }
          vehs[v].flag = 0;
      }
      else {
              if (zn) {
            Dx = vehs[v].dest[X] - vehs[v].band_wd;
            if ((Dx - dif) <= 0.0) {
                vehs[v].dest[X] = dif;
                    vehs[v].flag = 1;
            }
             else
                vehs[v].dest[X] = Dx;
          }
          else {
            Dx = vehs[v].dest[X] + vehs[v].band_wd;
            if ((Dx + dif) >= max_X_length) {
                vehs[v].dest[X] = max_X_length - dif;
                    vehs[v].flag = 1;
             }
             else
```

```
        }
        vehs[v].msg_idx = k;
   }
   else
        vehs[v].msg_idx = 0;
}

jd = vehs[v].turn % 4;

idx = vehs[v].id;
dif = (idx + 1) * vh_int;

switch (jd) {
  case 0:
        vehs[v].gwp_flg = 1;
        vehs[v].dest[Y] = max_Y_length;
        break;
  case 1:
        if (vehs[v].flag) {
           Dx = vehs[v].id * vh_int;
           if (zn) {
                vehs[v].dest[X] += Dx;
                vehs[v].dir = 0;
           }
           else {
                vehs[v].dest[X] -= Dx;
                vehs[v].dir = 1;
           }
           vehs[v].flag = 0;
        }
        else {
           if (zn) {
                Dx = vehs[v].dest[X] - vehs[v].band_wd;
                if ((Dx - dif) <= 0.0) {
                   vehs[v].dest[X] = dif;
                   vehs[v].flag = 1;
                }
                else {
                        vehs[v].dest[X] = Dx;
                }
           }
           else {
                Dx = vehs[v].dest[X] + vehs[v].band_wd;
                if ((Dx + dif) >= max_X_length) {
```

```
              vehs[v].turn++;
           }

           return(flg);
}


void get_newGWP(int v)
{
/*                                                        */
/* Compute a new Global Way Point for the vehicle  */
/*                                                               */
        int  jd, zn, idx;
        float Dx, Dy, dif;
           int i, k;
           struct com_buf tmp[5];

           zn = vehs[v].dir;

           if (vehs[v].msg_idx > 0) {
              k = 0;
              for (i=0; i<vehs[v].msg_idx; i++) {
                     if (vehs[v].msg[i].trn == vehs[v].turn) {
                        vehs[v].id--;
                        vehs[v].band_wd -= vh_int;
                     /*
                        if (zn)
                              vehs[v].dest[X] += vh_int;
              else
                              vehs[v].dest[X] -= vh_int;
                     */
                     }
                     else {
                        tmp[k].flt = vehs[v].msg[i].flt;
                        tmp[k].ids = vehs[v].msg[i].ids;
                        tmp[k].trn = vehs[v].msg[i].trn;
                        k++;
                     }
              }
              if (k > 0) {
                     for (i=0; i<k; i++) {
                        vehs[v].msg[i].flt = tmp[i].flt;
                        vehs[v].msg[i].ids = tmp[i].ids;
                        vehs[v].msg[i].trn = tmp[i].trn;
```

```
yp = vehs[v].dgps[Y];
   Dx = vehs[v].dest[X];
   Dy = vehs[v].dest[Y];
flg = 0;
   tol = 1.0;

jd = vehs[v].turn % 4;
   zn = vehs[v].dir;
   switch (jd) {
      case 0:
            if (yp+tol >= Dy) {
               flg = 1;
            }
            break;
      case 1:
            if (zn) {
               if (xp-tol <= Dx)
                        flg = 1;
            }
            else {
               if (xp+tol >= Dx)
                     flg = 1;
            }

      break;
      case 2:
            if (yp-tol <= Dy) {
         flg = 1;
      }
      break;
      default:
      if (zn) {
         if (xp-tol <= Dx)
            flg = 1;
      }
      else {
         if (xp+tol >= Dx)
            flg = 1;
      }

      break;
   }

   if (flg) {
```

```
/*                                                    */
/*    Compute the direction from the vehicles position to the target */
/*                                                    */
        float ang;
           float gpsx, gpsy, tmpy;
           float A, B;

           gpsx = vehs[v].dest[X];
           if (vehs[v].gwp_flg) {
              tmpy = 50.0;
              if (vehs[v].gwp_flg <= 5)
                    vehs[v].gwp_flg++;
              else
                    vehs[v].gwp_flg = 0;
              if ((vehs[v].turn % 4) == 0)
                    gpsy = vehs[v].dgps[Y] + tmpy;
              else
                    gpsy = vehs[v].dgps[Y] - tmpy;
           }
           else {
              gpsy = vehs[v].dest[Y];
           }

           A = gpsy - vehs[v].dgps[Y];
           B = gpsx - vehs[v].dgps[X];
        ang = atan2f(A, B);
           vehs[v].A0 = A;
           vehs[v].B0 = B;
        if (ang < 0.0)
               vehs[v].psi = ang + twopi;
        else
               vehs[v].psi = ang;
}


int chk_bnd_area(int v)
{
/* Check boundary area */
        float xp, yp;
        int flg, jd, zn;
           float Dx, Dy;
           float tol;

        xp = vehs[v].dgps[X];
```

```
                ch[7] = ch[7] + x;
        else if (x < 100) {
                dv = div(x,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }
        else {
                dv = div(x,100);
                ch[5] = ch[5] + dv.quot;
                y = dv.rem;
                dv = div(y,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }

           tsum = 0.0;
           wsum = 0.0;
           vsum = 0;
        inptr = fopen(ch,"w");
        for (j=0;j<ittr ;j++) {
               fprintf(inptr, "%5d %5d %5d", j+1, nv[j], tcl[j]);
               tmp = ((float)tcl[j]) / nv[j] * 100.0;
               tsum += tmp;
               vsum += ttm[j];
               tmp1 = *(av_map+j);
               wsum += tmp1;
            fprintf(inptr, "%7.2f %7.2f %5d %5d\n", tmp, tmp1, no_veh, ttm[j]);
        }

           tsum /= ittr;
           wsum /= ittr;

           fprintf(inptr, "%5d %5d %5d", 0, 0, 0);
           fprintf(inptr,   "%7.2f   %7.2f   %5d   %8.2f\n",   tsum,   wsum,   0,
(float)vsum/ittr);
        fclose(inptr);

           printf("The average searched space over all iterations is: %8.2f%%\% \n",
wsum);
        printf("\n");
    }

    void veh_dir(int v)
    {
          FILE *inptr;

          strcpy(ch, "fldva000.rst");
        x = no_veh;

        if (x < 10 )
                ch[7] = ch[7] + x;
        else if (x < 100) {
                dv = div(x,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }
        else {
                dv = div(x,100);
                ch[5] = ch[5] + dv.quot;
                y = dv.rem;
                dv = div(y,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }

          inptr = fopen(ch,"w");
        for (j=0; j<=field_y; j++) {
            for (i=0; i<=field_x; i++) {
                fprintf(inptr, "%3d", field[i][j]);
                }
               fprintf(inptr, "\n");
          }
}

void wr_cl_dt(int *nv, int *ttm, int *tcl, float *av_map, int ittr)
{
        div_t dv;
        char ch[13];
        int j, x, y, vsum;
        float tmp, tmp1, tsum, wsum;

        FILE *inptr;

        strcpy(ch, "pvacr000.rst");
        x = no_veh;

        if (x < 10 )
```

```
{
        div_t dv;
        char ch[13];
        int j, x, y;
        float tmp, tmp1;

        FILE *inptr;

        strcpy(ch, "panva000.rst");
        x = no_veh;

        if (x < 10 )
                ch[7] = ch[7] + x;
        else if (x < 100) {
                dv = div(x,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }
        else {
                dv = div(x,100);
                ch[5] = ch[5] + dv.quot;
                y = dv.rem;
                dv = div(y,10);
                ch[6] = ch[6] + dv.quot;
                ch[7] = ch[7] + dv.rem;
        }


        inptr = fopen(ch,"w");
        for (j=0;j<=cltb_ix ;j++) {
                tmp = ((float) cleartb[j]) / ((float) size);
                fprintf(inptr, "%10.2f \n", tmp);
        }
        fclose(inptr);
}

void wr_field()
/*                              */
/* print the map of the field       */
/*                              */
{
        int i, j, x, y;
           div_t dv;
        char ch[13];
        fac = sqrtf(-2.0*log(r)/r);

        iset = 1;
        gset = v1*fac;
        return (v2*fac);
    }
    else
    {
        iset = 0;
        return (gset);
    }
}

void wr_xy(void)
{
        int v;

        for (v=0; v<no_veh; v++)
                fprintf(outfx, "%12.3f", vehs[v].pnew[X] );
        fprintf(outfx, "\n");

        for (v=0; v<no_veh; v++)
                fprintf(outfy, "%12.3f", vehs[v].pnew[Y]);
        fprintf(outfy, "\n");
}

float avrg(int tm[], int nt[], int size)
{
        /* Compute the mean value of a list */

        int j;
        float sum, tmp;

        sum = 0.0;
        for (j=0; j<size; j++) {
                tmp = ((float) tm[j]) / ((float) nt[j]);
                sum += tmp;
        }

        return(sum / ((float) size)*100.0);
}

void wr_av_cl(int size)
```

```
{
/*-------------------------------------------------------------------- */
/*    Function:  gps_vh_err*/
/*    Parameters: */
/*    Returns: value |vehicle DGPS error| <= 1.0 */
/*-------------------------------------------------------------------- */

            return(rderr() * gps_err / 3.0);
}


float rderr()
/*-------------------------------------------------------------------- */
/*    Returns: value |vehicle DGPS error| <= 3.0 */
/*-------------------------------------------------------------------- */
{
        float err;
        do {
                err = randn();
        } while ( fabsf(err) > 3.0 );
       return(err);
}

float randn()
/*-------------------------------------------------------------------- */
/*    Function: randn */
/*    Summary: taken from gasdev() in Numerical Recipes in C */
/*    Parameters: */
/*    Returns: gauss distributed random value */
/*-------------------------------------------------------------------- */
{
        static int iset = 0;
        static float gset;

        float fac,r,v1,v2;

    if (!iset)
    {
        do
        {
                v1 = 2.0*(float) drand48() - 1.0;
                v2 = 2.0*(float) drand48() - 1.0;
                r = v1*v1 + v2*v2;
        } while ((r >= 1.0) || (r == 0.0));
```

```
                        vy = vehs[j].pnew[Y];
                        if ((vx >= tmx) && (vx <= tpx) && (vy >= tmy) && (vy <=
tpy)) {
                            tmp = (vx - tx) * (vx - tx) + (vy - ty) * (vy - ty);
                            dst = sqrtf(tmp);
                            if (dst <= rang) {
                                vehs[j].hm_fl = 3;
                                no_alive--;
                                no_destroy++;
                                msgtosup(j);
                            }
                        }
                    }
                }
            }
        }

        void msgtosup(int v)
        {
        /* Sends a message to the supervisor */
                int i;

                i = sup.idx;

                sup.buf[i].flt = vehs[v].fleet;
                sup.buf[i].ids = vehs[v].id;
                sup.buf[i].trn = vehs[v].turn;

                sup.idx++;
        }

        void move_bugs(int v)
        {
            float dx, dy;

            dx = vehs[v].speed * cos(vehs[v].psi);
            dy = vehs[v].speed * sin(vehs[v].psi);
            vehs[v].pnew[X] += dx;
            vehs[v].pnew[Y] += dy;
            vehs[v].dgps[X] = vehs[v].pnew[X] + vh_err();
            vehs[v].dgps[Y] = vehs[v].pnew[Y] + vh_err();
        }

        float vh_err()
```

```
                }
              }
          }

          *ly = y[1];
          *uy = y[2];
}


void dead_veh(int v)
/*                                                    */
/* The vehicle either exploded or the battery ran out        */
/*                                                    */
{
          int i, j;
          float tx, ty, tmx, tpx, tmy, tpy;
          float vx, vy, tmp, dst, rang;

          vehs[v].hm_fl = 3;
          no_alive--;
          no_cleared++;
          no_destroy++;
          msgtosup(v);

          i = vehs[v].explo;

          if (tgt[i].atr > 0) {
             tgt[i].sch_flg = 1;
             tx = tgt[i].pos[X];
             ty = tgt[i].pos[Y];

             if (tgt[i].atr == 1)
                   rang = 50.0;
             else
                   rang = 20.0;

             tmx = tx - rang;
             tpx = tx + rang;
             tmy = ty - rang;
             tpy = ty + rang;

             for (j=0; j<no_veh; j++) {
                   if (vehs[j].hm_fl != 3) {
                      vx = vehs[j].pnew[X];
```

```
        return(idx);
}

void intersect(float *PA, float *PB, float *PC, float xx, float *ly, float *uy)
/*                                  */
/* Find two intersect points at xx */
/* Return y values                  */
/*                                  */
{
        float a[4], y[4], inf, eps;
        int i, flg;
        int out, in;
        float tmp;


        inf = 1.0e10;
        eps = 1.0e-6;

        for (i=0; i<4; i++) {
           a[i] = - *(PA + i) * xx - *(PC + i);
        }

        flg = 1;
        for (i=0; i<4; i++) {
           if (fabsf(*(PB+i)) < eps) {
                if (flg) {
                   y[i] = -inf;
                   flg = 0;
                }
                else
                   y[i] = inf;
           }
           else
                y[i] = a[i] / *(PB+i);
        }

        for (out=0; out<3; out++) {
         for (in=out+1; in<4; in++) {
            if (y[out] > y[in])
            {
               tmp = y[in];
               y[in] = y[out];
               y[out] = tmp;
```

```
                minx = P[i][X];
            if (P[i][X] > maxx)
             maxx = P[i][X];
            if (P[i][Y] < miny)
             miny = P[i][Y];
            if (P[i][Y] > maxy)
             maxy = P[i][Y];
        }

        for (i=0; i<no_target; i++) {
            if (tgt[i].sch_flg)
                continue;
            px = tgt[i].pos[X];
            py = tgt[i].pos[Y];
            if ((px >= minx) && (px <= maxx) && (py >= miny) && (py <= maxy))
{
                intersect(A, B, C, px, &ity1, &ity2);
                if ((py >= ity1) && (py <= ity2)) {
                    idx = i;
                    break;
                }
            }
        }

        /* Mark searched field */

        lx = minx + 1;
        if ( lx < 0) lx = 0;
        ux = maxx;
        if (ux > field_x) ux = field_x;

        for (i=lx; i<=ux; i++) {
            px = (float)i;
            intersect(A, B, C, px, &ity1, &ity2);
            ly = ity1 + 1;
            if (ly < 0) ly = 0;
            uy = ity2;
            if (uy > field_y) uy = field_y;

            for ( j=ly; j<=uy; j++) {
                if (!(field[i][j]))
                    field[i][j] = 1;
            }
        }
```

```
int   srh_tg(int v)
{
        int idx, i, j;
        int lx, ux, ly, uy;
        float A[4], B[4], C[4], P[4][XY];
        float x0, y0;
        float dx, dy, thc, ths, x1, y1;
        float ity1, ity2, px, py;
        float minx, maxx, miny, maxy;

        idx = -1;

        thc = cos(vehs[v].psi);
        ths = sin(vehs[v].psi);
        x0 = vehs[v].pnew[X];
        y0 = vehs[v].pnew[Y];
        x1 = x0 + vehs[v].speed * thc;
      y1 = y0 + vehs[v].speed * ths;
        A[0] = A[1] = vehs[v].B0;
        B[0] = B[1] = vehs[v].A0;
        C[0] = - A[0] * x0 - B[0] * y0;
        C[1] = - A[1] * x1 - B[1] * y1;
        dx = sensorWd2 * ths;
        dy = sensorWd2 * thc;
        P[0][X] = x0 - dx;
        P[0][Y] = y0 + dy;
        P[1][X] = x0 + dx;
        P[1][Y] = y0 - dy;
        P[2][X] = x1 - dx;
        P[2][Y] = y1 + dy;
        P[3][X] = x1 + dx;
        P[3][Y] = y1 - dy;
        A[2] = A[3] = vehs[v].A0;
        B[2] = B[3] = - vehs[v].B0;
        C[2] = - A[2] * P[0][X] - B[2] * P[0][Y];
      C[3] = - A[3] * P[1][X] - B[3] * P[1][Y];

        /* find minimum and maximum values */

        minx = maxx = P[0][X];
        miny = maxy = P[0][Y];
        for (i=1; i<4; i++) {
           if (P[i][X] < minx)
```

```
void ch_idx(int v, int ix)
/*                          */
/* Change the vehicle id & GWP  */
/*                          */
{
        int i;

        /* Change the GWP for the vehicle's id < v */

        i = vehs[v].buf[ix].ids;

        vehs[v].band_wd -= vh_int;
        if (vehs[v].id < i) {
            if (!(vehs[v].flag))
                ch_GWP(v);
        }
        else {
                /* Change the vehicle ID > v    */
            vehs[v].id--;
        }
}

void ch_GWP(int v)
{
/*                          */
/* Change the Global Way Point   */
/*                          */

        int  jd, zn;

        jd = vehs[v].turn % 4;
        zn = vehs[v].dir;

        if (zn)
            vehs[v].dest[X] += vh_int;
        else
            vehs[v].dest[X] -= vh_int;

        if ((jd == 0) || (jd == 2)) {
            vehs[v].gwp_flg = 1;
            veh_dir(v);
        }
}
```

```
                    veh_dir(v);
                         vehs[v].hm_fl = 0;
                      }

        else {
                         if ((vehs[v].clock % tr_head_ch) == 0) {
                              veh_dir(v);
                    }
                      }
                      break;
           case 2:
                      vehs[v].wait_tm--;
                      if (vehs[v].wait_tm <= 0) {
                         /* Explosion */
                         dead_veh(v);
                      }
                      break;
           case 3:
                      /* vehicle dead */
                      break;
         default:
            break;
      }
}

void getmsg(int v)
{
/* receive the message sent from the supervisor */
        int i, j;

        for (i=0; i<vehs[v].buf_idx; i++) {
          if (vehs[v].turn >= vehs[v].buf[i].trn)
                 ch_idx(v, i);
           else {
                 j = vehs[v].msg_idx;
                 vehs[v].msg[j].flt = vehs[v].buf[i].flt;
                 vehs[v].msg[j].ids = vehs[v].buf[i].ids;
                 vehs[v].msg[j].trn = vehs[v].buf[i].trn;
                 vehs[v].msg_idx++;
           }
        }

        vehs[v].buf_idx = 0;
}
```

```
                    vehs[i].dir    = 1;
                    vehs[i].id      = vm1 - i;
                    vehs[i].band_wd = wd[1];
                    vehs[i].fleet   = 1;
                    vehs[i].clock   = 0 - (vdiv2 + vehs[i].id) * delay;
                  }
               }

                    wr_rst(i);
          }

            no_alive = no_veh;
            no_destroy = 0;
}

void move_one_sec(int v)
{
        int flag, index, idx;

            if (vehs[v].hm_fl == 1) {
               if (vehs[v].buf_idx > 0)
                    getmsg(v);

                    /*  Search target */
            index = srh_tg(v);
            if (index >= 0) {
                    vehs[v].hm_fl = 2;
                    vehs[v].explo = index;
                    vehs[v].wait_tm = Bch_tm;
               }
               else
                    move_bugs(v);
            }

            switch (vehs[v].hm_fl) {
               case 0:
                    vehs[v].wait_dch--;
                    if (vehs[v].wait_dch <= 0)
                       vehs[v].hm_fl = 1;
                    break;
               case 1:
                    if (chk_bnd_area(v) ) {
                       get_newGWP(v);
                  vehs[v].wait_dch = dir_hch;
```

```
   else
        tmp_x += (float)st_x + 1.0;

for ( i=0; i<no_veh; i++)
{
        vehs[i].pnew[X] = tmp_x;
        vehs[i].pnew[Y] = tmp_y;
        vehs[i].dgps[X] = tmp_x + vh_err();
        vehs[i].dgps[Y] = tmp_y + vh_err();
        vehs[i].dest[X] = tmp_x;
        vehs[i].dest[Y] = max_Y_length;
        tmp_x          -= vh_int;
        vehs[i].turn    = 0;
        vehs[i].flag    = 0;
        vehs[i].speed   = vh_tr_spd;
        vehs[i].hm_fl   = 0;
            vehs[i].buf_idx = 0;
            vehs[i].msg_idx = 0;
        veh_dir(i);
        if (flg) {
           if (i <= vdiv2) {
              vehs[i].dir    = 0;
              vehs[i].id      = i;
              vehs[i].band_wd = wd[0];
              vehs[i].fleet   = 0;
              vehs[i].clock   = 0 - i * delay;
           }
           else {
              vehs[i].dir    = 1;
              vehs[i].id      = vm1 - i;
              vehs[i].band_wd = wd[1];
              vehs[i].fleet   = 1;
              vehs[i].clock   = 0 - (vdi2p1 + vehs[i].id) * delay;
           }
        }
        else {
           if (i < vdiv2) {
              vehs[i].dir    = 0;
              vehs[i].id      = i;
              vehs[i].band_wd = wd[0];
              vehs[i].fleet   = 0;
              vehs[i].clock   = 0 - i * delay;
           }
           else {
```

```
        printf("vehs[i].pnew[X] = %f \n", vehs[i].pnew[X]);
        printf("vehs[i].pnew[Y] = %f \n", vehs[i].pnew[Y]);
        printf("vehs[i].dgps[X] = %f \n", vehs[i].dgps[X]);
        printf("vehs[i].dgps[Y] = %f \n", vehs[i].dgps[Y]);
        printf("vehs[i].speed   = %f \n", vehs[i].speed);
        printf("vehs[i].hm_fl   = %d \n", vehs[i].hm_fl);
        printf("vehs[i].clock   = %d \n", vehs[i].clock);
          printf("vehs[i].band_wd = %f \n", vehs[i].band_wd);
}


void init_veh()
/*------------------------------------------------------------------- */
/*    Function:  init_veh                                          */
/*    Parameters:                                                 */
/*    set the initial position for each vehicle                      */
/*    Get the target for searching and define searching area              */
/*------------------------------------------------------------------- */
{
        float tmp_x, tmp_y;
        int i, st_x, vdiv2, vdi2p1, flg, vm1;
        float wd[2];

          tmp_y = 0.0;
        vm1 = no_veh - 1;

        st_x = max_X_length / 2.0;
        vdiv2 = no_veh / 2;
        vdi2p1 = vdiv2 + 1;

        tmp_x = vh_int * vdiv2;
        wd[1] = tmp_x;

        if (no_veh % 2) {
           wd[0] = vh_int + tmp_x;
           flg = 1;
        }
        else {
           wd[0] = tmp_x;
           flg = 0;
        }

          if (vh_int > 3.95 && vh_int < 4.05)
           tmp_x += (float)st_x;
```

```
                        vehs[j].buf_idx++;
                    }
                 }
             }
          }
          sup.idx = 0;
}

float find_avmap()

/*                                                    */
/* find an average of the searched area in the minefield  */
/*                                                    */
{
       int i, j, tot;

          tot = 0;
          for (i=0; i<=field_x; i++)
          for (j=0; j<=field_y; j++) {
              if (field[i][j] == 0 )
                       tot++;
          }

          return((1.0 - (float)tot / ((float)(field_x+1) * (float)(field_y+1))) * 100.0);
}


void init_field()
/*                              */
/* Initialize the test field          */
/*                              */
{
          int i, j;

          for (i=0; i<=field_x; i++)
             for (j=0; j<=field_y; j++)
                   field[i][j] = 0;
}

void wr_rst(int i)
{
       printf("vehs = %d \n", i);
          printf("vehs id = %d \n", vehs[i].id);
          printf("fleet   = %d \n", vehs[i].fleet);
```

```
                        cleartb[tmp_ix]  = cleartb[dv.quot];
                        cltb_ix = tmp_ix;
                    }
                }
            }
            cltb_last = cleartb[cltb_ix];

            tot_dead[itr-1] = no_destroy;
            tot_cled[itr-1] = no_cleared;
               no_tg[itr-1] = no_target;

               av_map[itr-1] = find_avmap();

            printf("\n");
               printf("The average searched area in iteration %d is:         %8.2f%% \n",
itr, av_map[itr-1]);
               printf("The number of swimmers destroyed in iteration %d is:  %5d \n",
itr, no_destroy);
            printf("%5d targets of %5d cleared in %7d steps \n", no_cleared,
no_target, step);
        }

        printf("\n");
        printf("The average percent of cleared targets over all iterations is:
%10.3f%% \n", avrg(tot_cled, no_tg, no_itr));

        wr_av_cl(no_itr);
        wr_cl_dt(no_tg, tot_dead, tot_cled, av_map, no_itr);
    }

    void supervisor()
    {
    /*      communicate with the vehicles     */

            int i, j, k;

            for (i=0; i<sup.idx; i++) {
               for (j=0; j<no_veh; j++) {
                     if (vehs[j].hm_fl != 3) {
                        if (vehs[j].fleet == sup.buf[i].flt) {
                              k = vehs[j].buf_idx;
                              vehs[j].buf[k].flt = sup.buf[i].flt;
                              vehs[j].buf[k].ids = sup.buf[i].ids;
                              vehs[j].buf[k].trn = sup.buf[i].trn;
            if (dv.quot <= cltb_ix) {
                cleartb[dv.quot] += no_cleared;
                       }
            else {
                cleartb[dv.quot] = cltb_last + no_cleared;
                       }
        }
    }

    step++;
}

if (itr == no_itr) {
    fclose(outfx);
    fclose(outfy);
           wr_field();
}

dv = div(step-1, 60);
tmp_ix = dv.quot+1;
if (dv.rem == 0) {
    if (itr == 1)
        cltb_ix = dv.quot;
    else {
        if (dv.quot < cltb_ix)
            for (i=tmp_ix; i<= cltb_ix; i++) {
                cleartb[i] += no_cleared;
                       }
        else
            cltb_ix = dv.quot;
    }
}
else {
    if (itr == 1) {
        cleartb[tmp_ix] = no_cleared;
        cltb_ix = tmp_ix;
    }
    else {
        if (tmp_ix <= cltb_ix) {
            for (i=tmp_ix; i<= cltb_ix; i++) {
                cleartb[i] += no_cleared;
                       }
        }
        else {
```

```
   /* wr_rst(); */

if (itr == no_itr) {
    outfx = fopen("x.out","w");
    outfy = fopen("y.out","w");
    wr_xy();
}

no_cleared = 0;
done = 0;
step = 1;

while(!done) {

    for (veh=0; veh<no_veh; veh++) {

        vehs[veh].clock++;

        if (vehs[veh].clock <= 0)
            continue;

        move_one_sec(veh);

        if ((vehs[veh].hm_fl < 3) && (vehs[veh].clock >= no_step)) {
                        vehs[veh].hm_fl = 3;
                        no_alive--;
        }
    }

           if (sup.idx > 0)
               supervisor();

    if (itr == no_itr)
            wr_xy();

    if (no_alive <= 0)
            done = 1;

    dv = div(step, 60);
    if (dv.rem == 0) {
        if (itr == 1) {
            cleartb[dv.quot] = no_cleared;
               }
        else {
                }

                   printf("\n");

                cleartb[0] = 0;

                   printf("Input the desired vehicle speed: (in knots) \n");
                   scanf("%f", &veh_nm);
                printf("\n");

                   printf("Input the desired sensor range of the vehicle: (in meters) \n");
                   scanf("%f", &veh_swath);
                   printf("\n");

                set_env();

                   printf("Input the number of seconds you want to run the simulation for.
\n");
                printf("The maximum time the program will run is %d seconds. \n",
max_step);
                printf("An input value of zero will default the simulation to maximum. \n");
                scanf("%d", &no_step);

                   if (no_step == 0)
                           no_step = max_step;

                printf("\n");

                for (itr=1; itr<=no_itr; itr++)
                {

                           /* Initialize the targets   */
                      init_target();

                           /* Initialize swimmers     */
                   init_veh();

                           /* Initialize fields           */

                      init_field();

                           /* initialize supervisor   */
                      sup.idx = 0;
```

```
            }

          delay = B1_dt / vh_tr_spd + Bch_tm + 1;
          printf("\n");
          printf("The delay time between vehicles starting is:  %d sec \n", delay);

          tr_head_ch = 7;
            tg_pcc     = 1.0;

          max_step = 21600;

            printf("\n");
     }

     main(argc, argv)
     int argc;
     char *argv[];
     {
          int no_itr;
          int itr;
          int done, i;
          int stepm1, veh, no_step, tmp_ix, cltb_last;
            float clOb_last;
          div_t dv;
            int tot_dead[1000]; /* Total # of dead vehicles for each iteration */
            int no_tg[1000];    /* # of targets for each iteration               */
          int tot_cled[1000]; /* Total # of cleared targets for each iteration */
            float av_map[1000];

          if (argc != 3) {
            fprintf(stderr, "Usage: <program name> <# of vehicles> <# of
iterations>\n");
            return 1;
          }

          no_veh = atoi(argv[1]);
          printf("\n");
          printf("The number of vehicles for this simulation is:   %d \n", no_veh);
          no_itr = atoi(argv[2]);
          printf("The number of iterations for this simulation is: %d \n", no_itr);

          if (!(vehs = (struct swimmer *)malloc(no_veh * sizeof(struct swimmer)))) {
               fprintf(stderr, "bounce: malloc failed\n");
               return 1;
```

```
           max_X_length = 165 * YD;           /* 165 Yards        */
           max_Y_length = (10330 - 1259) * FT;
              field_x = max_X_length;
              field_y = max_Y_length;

              printf("The simulation field is: \n");
              printf("    %8.2f m by %8.2f m \n", max_X_length, max_Y_length);
              printf("\n");

              /* Establishment of the vehicle characteristics */
           vh_tr_spd  = veh_nm * NM / 3600.0;  /* vehicle speed in m/sec           */
              tg_sensorW = veh_swath;        /* Target sensor influence width      */
           dir_hch   = 1;                     /* Vehicle turning time 2 sec        */
              gps_err   = 2.0;                  /* 2.0 m of GPS Error                */
              tg_pos_err = 1.0;                  /* |tg_pos_err| <= 1.0           */
              B1_dt      = 50.0;
              B2_dt      = 20.0;
              Bch_tm     = 15;

              printf("The vehicle characteristics for this simulation are: \n");
              printf("     Speed is:           %f m/sec \n", vh_tr_spd);
              printf("     Sensor range is:    %f m \n", tg_sensorW);
              printf("     GPS error is:       %f m \n", gps_err);
              printf("     Charge time is:     %d sec \n", Bch_tm);
           printf("\n");

                  /* computation of vehicle spacing    */
              sensorWd2 = tg_sensorW / 2.0;
              vh_int  = sensorWd2;  /* establishes the default value   */

      input_phase:
              printf("The spacing between vehicles is %f m. \n", vh_int);
              printf("         percent  overlap  is:  %7.2f%%  \n",  (tg_sensorW  -  vh_int)  /
vh_int*100.0);
           printf("\n");
              ch = getchar();
              printf("Is this the desired overlap? (y or n) \n");
              ch = getchar();
           if (ch != 'y') {
               printf("\n");
                  printf("Enter the desired vehicle interval: (in meters) \n");
                  scanf("%f", &vh_int);
               printf("\n");
                  goto input_phase;
```

```
float pos_err( );
float randn(void);
float avrg(int[], int[], int);
void  wr_av_cl(int);
int   chk_bnd_area(int);
float rand_pi(int);
void  veh_dir(int );
void  wr_rst(int);
void  dead_veh( int);
void  get_newGWP(int);
void  ch_GWP(int);
void  wr_cl_dt(int *, int *, int *, float *, int );
void  wr_field(void);
float find_avmap(void);
void  intersect(float *, float *, float *, float, float *, float *);
void ch_idx(int, int);
void msgtosup(int);
void supervisor();
void getmsg(int);

void set_env(void)
{
/* Definition of the vehicle characteristics */
/* and search field environment.            */
/*                                          */
        int tmp;
        int j;
        float tmp1;
           float FT, YD, NM;
           char ch;
        int intv;

           FT = 0.3048;   /* 1 foot = 0.3048 meters          */
           YD = 3.0 * FT;          /* 1 yard = 3 foot                */
        pi = 4.0 * atanf(1.0);
           NM = 1852.0; /* 1 nautical mile = 1852 meters     */

        twopi = 2.0 * pi;
        thrpi = twopi + pi;
        thpi =  thrpi / 2.0;
        piov2 = pi / 2.0;
        piov4 = piov2 / 2.0;

           /*  Establishment of the search field environment */
        float tg_dst;    /* Target sensor distance                */
           int explo;       /* Searched target index                      */
        int wait_dch;    /* Heading change time                    */
           int wait_tm;     /* Vehicle waiting time                  */
        int clock;       /* Vehicle start time from home          */
           float band_wd;   /* Search band width                      */
           int buf_idx;      /* Message buffer                              */
           struct com_buf buf[5];
           int msg_idx;      /* Saved message                               */
           struct com_buf msg[5];
        };
struct swimmer *vehs;

struct target {
        float pos[XY]; /* Target position                              */
        float gps[XY]; /* DGPS of target                               */
           int   atr;     /* Target attribute                          */
                     /* if = 0 : false target                        */
                     /* if = 1 : Bottom Influence @ 50 meter spacing */
                     /* if = 2 : Moored Contact @ 20 m spacing       */
        int sch_flg;   /* if = 0 : target is unknown                 */
                     /* if = 1 : target was found and cleared        */
        };
struct target tgt[max_tg];

int cleartb[800];      /* contains total # of cleared targets   */
int cltb_ix;           /* index of clear table                  */

int   field_x, field_y;
short int field[151][2765];

FILE *outfx, *outfy;

void  set_env(void);
void  init_veh(void);
void  init_target(void);
void  init_field(void);
void  move_one_sec(int);
void  chk_tg_po(int, int);
int   srh_tg(int);
void  wr_xy(void);
void  move_bugs(int);
float rderr(void);
float vh_err();
```

```
int   Bch_tm;       /* Charging time to destroy mine           */
int   delay;           /* Delay time between vehicle starts        */
float vh_int;       /* Space between vehicle to vehicle        */
int   comm_tm;               /* Vehicle Communication time                */
int   tr_head_ch;   /* random search heading change interval    */
float veh_nm;
float veh_swath;

struct com_buf {
          int flt;  /* No Fleet      */
          int ids; /* Vehicle id    */
          int trn; /* Vehicle turn*/
};


struct super {
          int idx;
          struct com_buf buf[10];
};

struct super sup;

struct swimmer {
          int   fleet;
          int   id;        /* Vehicle id                                     */
       float pnew[XY];  /* New position                           */
       float dgps[XY];  /* DGPS position                          */
          float dest[XY];  /* GWP                                                  */
          int   gwp_flg;   /* Path change flag                          */
       int   turn;      /* if turn = 0, move up                  */
                    /*         = 1  move right               */
                    /*         = 2  move down                */
                    /*         = 3  move right               */
          int   dir;       /* If dir = 0 Left to Right                   */
                             /*        = 1 Right to Left                    */
          int flag;
       float speed;     /* vehicles speed                        */
       float psi;       /* vehicles direction                    */
                             /* if hm_fl = 0 then wait                    */
          float A0;        /* A0 x + B0 y + C0 = 0 at pnew[XY]   */
          float B0;
       int hm_fl;       /* if hm_fl = 1 then navigate to GWP     */
                    /*         = 2 then target detonation   */
                    /*         = 3 then stop                */
```

fry31_rev_final.c

```
/*                                                        */
/* Simulation of multiple swimmer vehicle           */
/* robotic minesweeping in shallow water areas   */
/* using a lawnmower search pattern with            */
/* intermediate Global Way Points and with        */
/* a communication link between the vehicle        */
/* and a supervisor vehicle.                             */
/* Written by Joung K. Kim                              */
/* November 09 1999                                     */
/* Version 3                                               */
/* Modified by LT Peter M. Ludwig, USN          */
/* February-May 2000                                    */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define X       0
#define Y       1
#define XY      2
#define max_tg  20   /* Maximum No. of Targets  */

float pi, twopi, thrpi, thpi, piov2, piov4, turn_ang;

int no_veh;              /* Number of Vehicles       */
int no_alive;                /* Number of Vehicles alive */
int no_target;           /* Number of Targets        */
int no_destroy;
int no_cleared;

float max_X_length, max_Y_length;
float vh_tr_spd;    /* vehicle speed on transition                   */
float tg_sensorW;   /* Width of Target sensor                               */
float sensorWd2;    /* half of Width of Target sensor           */
int   step;         /* Time step                        */
int   dir_hch;      /* Heading change time (2 sec)              */
int   max_step;     /* Maximum allowable time step             */
float tg_pcc;       /* Probability of correct classification         */
                         /* of Target                                      */
float gps_err;      /* Vehicle GPS error                            */
float tg_pos_err;   /* Target position error when it placed     */
float B1_dt;        /* Mine #1 influence distance               */
float B2_dt;        /* Mine #2 influence distance              */
```

APPENDIX A. CODED PROGRAM FILE

This appendix contains the code for the computer simulations, written utilizing C programming language.

operations. Certainly as the myriad of technologies supporting this concept grows, so do the possibilities.

B. RECOMMENDATIONS

In support of the significant growth potential of this research and to further validate the feasibility and functionality of this concept, the following recommendations for additional study are made:

- Include more detailed swimmer guidance laws, including cross track and long track error control as opposed to the simple line of sight guidance used in this study.

- Account for already sensed mines escaping detonation and remaining active, such as would be seen with smart mines like ship counters or due to a misfire during the detonation process by a swimmer.

- Develop and utilize optimization techniques based on probability of complete clearance to analyze simulation results and determine the vehicle parameters most conducive to conducting robotic minesweeping operations.

V. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSION

The problem of robotic minesweeping is extremely complex, and must allow for multiple scenarios. Accordingly, due to the preliminary nature of this research, its scope was limited to verification of the validity of the concept. This was accomplished through detailed analysis of simulation code, improving the code, and conducting simulations to verify the results.

During the process of analyzing the code, four topics were identified as primary areas of concentration. These included the principle of overlap, effects of varying the turn time, vehicle identification procedures, and track re-allocation. Of these four areas, one, the effects of turn time, was found to be not as critical as previously thought.

Specific improvements to the code were made in the areas concerning calculation of overlap and track re-allocation. Additionally, minor changes were made throughout the code to help make it a more useful tool.

Simulations were conducted to verify the concept. While not the specific goal of the simulations, ballpark figures for the optimization of sensor width and vehicle spacing were uncovered. Additionally, without taking into consideration any of the resistive forces, certain limitations of using slower speed vehicles for the operations were shown.

Overall, this study provided extremely encouraging results and validated the concept of multi-vehicle fleets of robotic swimming vehicles being used for minesweeping

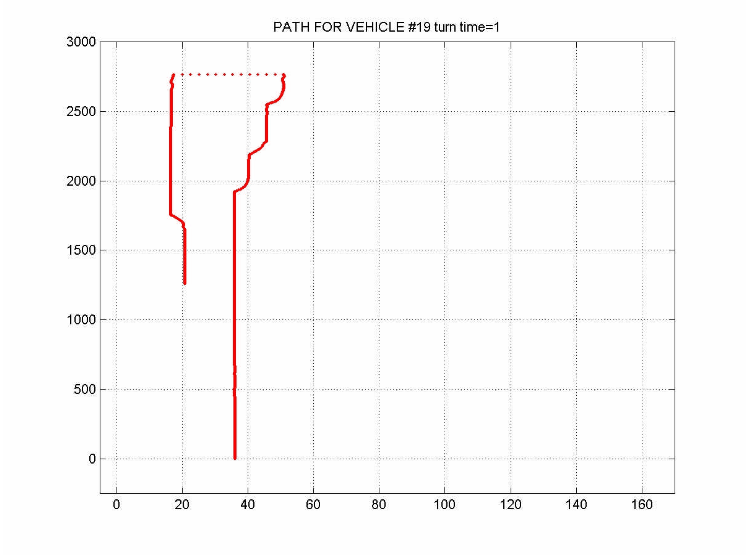


Figure 4.4 Rapid Sequence of Vehicle Re-tasking

Additionally, a unique situation was seen as depicted in Figure 4.5 where a mine was sensed and neutralized while a vehicle was being re-allocated. This displays the flexibility and rapid response to change that would be required of the vehicles.

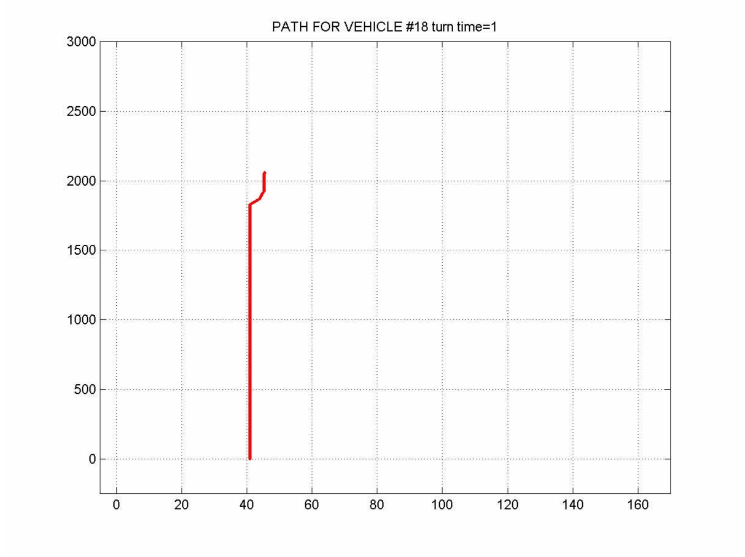


Figure 4.5 Mine Neutralization During Re-allocation

Figure 4.3 displays the converse. It shows the vehicles being spaced too far apart and therefore covering the entire minefield, but a large number of holidays in the areas covered account for the poor results seen with five meter spacing.

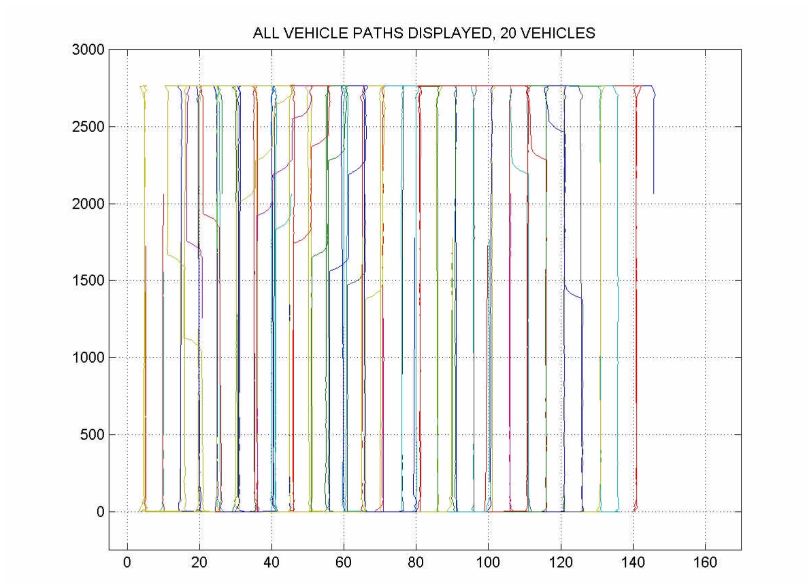


Figure 4.3 Simulated Tracks at 5 Knots and 5 m Spacing

Obviously, from the results seen the optimum value for vehicle spacing lies somewhere in the mid-range area.

D. INTERESTING RESULTS

During the course of reviewing the graphical outputs from the simulations conducted, a couple of interesting items were noticed. The first, as shown in Figure 4.4, is a fairly rapid sequence of vehicle re-tasking. Such a scenario is highly probable in reality, so it was interesting to note that the simulation handled it well.

in the coverage area would occur. However as seen in the data and as graphically depicted in Figure 4.2, a point of diminishing return can be reached and exceeded.

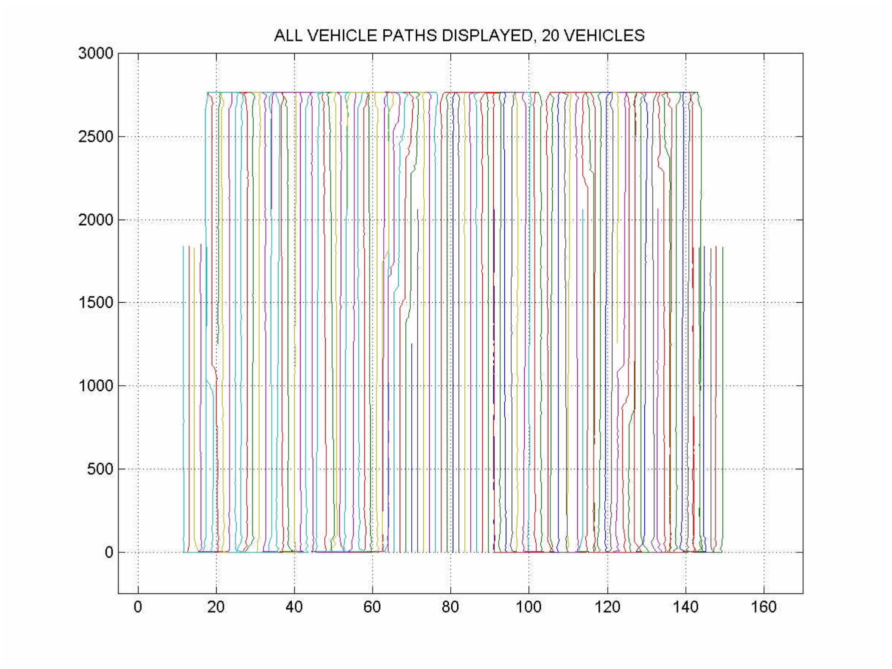


Figure 4.2 Simulated Tracks at 5 Knots and 1.5 m Spacing

Clearly seen in the figure, the vehicles are spaced to closely together so that the vehicles are unable to cover all the minefield before the simulation time runs out. While extremely good coverage and a high probability of successful detection of all mines is seen in the areas searched, the outer edges of the minefield were not searched and the overall results are poor. However, this also prompts the idea that perhaps narrow vehicle spacing while not performing well in larger minefields, might be much more aptly suited to use in areas where only a relatively narrow path need be swept, for example, a shipping lane.

C. EFFECTS OF VARYING THE VEHICLE INTERVAL

Table 4.5 reflects those simulation parameters that were maintained constant while determining the effects of varying the vehicle interval.

# of Vehicles	# of Iterations	Vehicle Speed	Sensor Width	Simulation Time
20	100	5 KTS	4.2672 m (14 ft)	7200 sec

Table 4.5 Simulation Parameters for Analysis of Varying Vehicle Spacing

For this group of simulations, the desire to see results for a very large, a mid-range, and a negative percent overlap fueled the decision of appropriate vehicle spacing numbers. Respectively, vehicle spacing of 1.5 m, 2.5 m, and 5 m were chosen. The tabulated results appear below in Table 4.6.

Vehicle Spacing	% Overlap	% Cleared Targets	% Searched Space
1.5 m (4.9213 ft)	184.48 %	87.055 %	91.63 %
2.5 m (8.2020 ft)	70.69 %	99.545 %	98.86 %
5 m (16.4042 ft)	-14.66 %	91.500 %	92.18 %

Table 4.6 Simulation Results for Various Vehicle Spacing

The results seen here provide good insight into the vehicle behavior. Initially, one would think that a greater percent overlap would likely yield better results as less holidays

# of Vehicles	# of Iterations	Vehicle Speed	Vehicle Spacing	Simulation Time
20	100	5 KTS	2.5 m (8.2020 ft)	7200 sec

Table 4.3 Simulation Parameters for the Analysis of Varying Sensor Width

For these simulations, sensor widths were chosen based on the length of a whisker arm. The initial choice of a ten foot swath width came from the minimum length of five feet specified for a whisker pole in the concept. The subsequent 14 foot and 18 foot runs resulted from incrementally adding an additional two feet to each whisker pole. The averaged results for all one hundred iterations of these runs are shown in Table 4.4.

Sensor Width	% Overlap	% Cleared Targets	% Searched Space
3.048 m (10 ft)	21.92 %	98.982 %	97.11 %
4.2672 m (14 ft)	70.69 %	99.545 %	98.86 %
5.4864 m (18 ft)	119.46 %	100.00 %	99.69 %

Table 4.4 Simulation Results for Various Sensor Widths

The results returned from these runs are extremely encouraging, especially the 100% average clearance rate seen for the 18 foot swath width. Considering that a very good percent clearance rate was also achieved for a 14 foot sensor width, the data clearly reveals that the optimal sensor width for the specified parameters lies somewhere between 14 feet and 18 feet.

knots. Figure 4.1 graphically shows the results of all the vehicles for the 2.5 knot simulation.

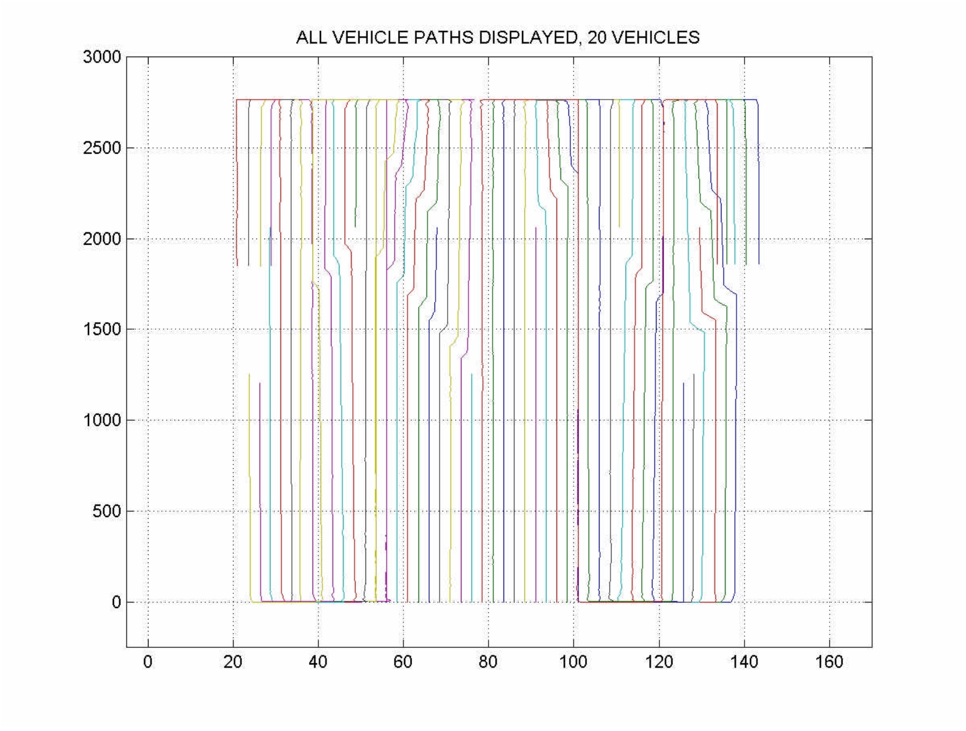


Figure 4.1 Simulated Tracks at 2.5 Knots

Clearly, because of the slow speeds, the vehicles were unable to reach the boundaries of the minefield in the allotted timeframe.

B. EFFECTS OF VARYING THE SENSOR WIDTH

The parameters maintained constant for the simulations where sensor width variations were considered are shown in Table 4.3.

# of Vehicles	**# of Iterations**	**Sensor Width**	**Vehicle Spacing**	**% Overlap**	**Simulation Time**
20	100	4.2672 m (14 ft)	2.5 m (8.2020 ft)	70.69 %	7200 sec

Table 4.1 Simulation Parameters for Analysis of Varying Vehicle Speeds

The results obtained for simulations using vehicle speeds of 2.5 KTS, 5 KTS, and 7 KTS are shown here in Table 4.2. It was anticipated that as the vehicles speed decreased, the simulation results would also show a definite decline in effectiveness. Certainly, these results prove this to be true. The results depicted are the average of the results for all one hundred iterations.

Vehicle Speed	**% Cleared Targets**	**% Searched Space**
2.5 KTS	88.382 %	80.77 %
5 KTS	99.545 %	98.86 %
7 KTS	99.727 %	99.06 %

Table 4.2 Simulation Results for Various Vehicle Speeds

Only clearing eighty-eight percent of the mines laid and searching only eighty-one percent of the minefield in the 2.5 knot run is clearly an unsatisfactory solution for a slow speed search. The decreasing effectiveness seen here could become a potential problem as vehicles start to encounter resistance forces such as drag forces, waves and ocean currents. Therefore, it must be recognized when designing the vehicles that their propulsion systems must possess enough power to be able to overcome these external effects and still achieve a reasonable speed over the ground, likely somewhere around five

IV. RESULTS

Of course, no conceptual study would be complete without a verification of the methods. To do this, it was decided to run multiple simulations and over the course of these simulations to alter the speed of the vehicle, vary the sensor width, and adjust the vehicle spacing. Although optimal results for the simulation would be found by altering multiple parameters simultaneously, due to the preliminary nature of this work, it was decided to change only one parameter at a time so that the effects of that change could be clearly understood.

For the purposes of these multiple simulation runs, only three parameters, the number of vehicles, the number of iterations, and the simulation time remained constant throughout all the simulations. Twenty vehicles were chosen for the runs to ensure that there would always be a greater number of vehicles than there would be mines. One hundred iterations were chosen to verify both the accuracy and precision of the results. The simulation time of 7200 seconds, or two hours, was chosen to represent the current technological limitation imposed on this concept by modeling an easily accomplished approximate endurance.

A. EFFECTS OF VARYING THE VEHICLE SPEED

The parameters shown in Table 4.1 are the parameters maintained constant while the vehicle speeds were altered.

offset (ÄX) from the position of the first vehicle, which is considered to be a reference point for the other vehicles.

2. Calculation of and usage of ÄX

The idea of vehicle ID re-assignment is the critical factor in determining a vehicle's desired horizontal offset. The other critical assumption is that the vehicles are on track and have not deviated beyond the limits of navigational error. In reality, this may be a problem due to varying factors such as ocean current forces, vehicle drag forces, and navigational system reliability, however, for the purposes of this research, these factors were ignored. So, under this assumption, simply knowing that the spacing between vehicles is to remain constant, coupled with the concept of vehicle ID reassignment, the current delta of the vehicle is given by

$$\Delta\text{X (i)} = \text{vehicle id (i)} \times \text{vehicle spacing.}$$

Once ÄX is calculated, the program enters the data into a series of conditional loops. Inside of these loops, ÄX is compared with the position of the desired GWP. The final result being the updated track, as seen in Figure 3.4, that takes the swimmer vehicle to the point where it can resume its nearly vertical path to its newly assigned GWP. Upon reaching the GWP, the swimmer using its updated GWP file continues with its regular search pattern until the simulation ends or another mine is sensed.

disseminated to the swimming vehicles. As an example, Figure 3.4 clearly shows such a shift.

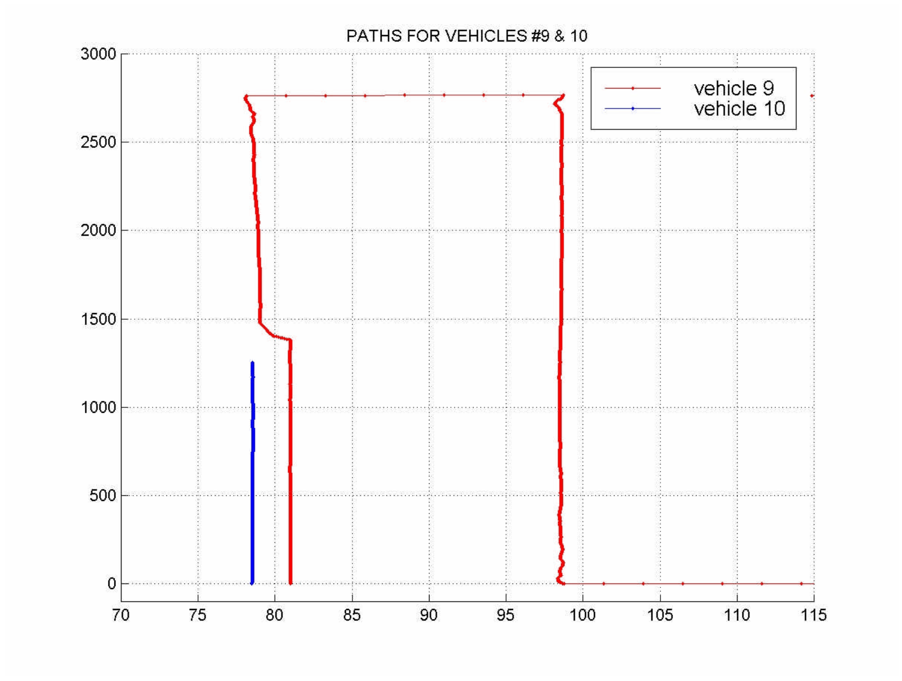


Figure 3.4 Example Vehicle Shift

As seen in this example, vehicle 10 was clearly lost to a mine in an area that correlates to the first mine danger area. When this occurred, vehicle 9, as directed by the supervisor, obviously alters course in order to shift its track to head for the newly assigned GWP that was previously allocated to vehicle 10. This allows the vehicles to search and sweep the maximum amount of area possible, leaving only minimal gaps of un-covered area (holidays) in the process. Other vehicles with ID numbers less than nine follow suit, but were left out of the figure for clarity.

Logically, determination of the course change required for the track re-allocation is a function of the vehicle's position. The vehicles position is calculated as a horizontal always referenced as zero. However, in order to try to reduce the confusion that this practice might present when viewing graphical displays of the data, the permanent vehicle ID correlates directly to its position in line. For example, the first vehicle in line is referred to as vehicle one and the twentieth vehicle in line is vehicle twenty. Initially, both the primary ID and secondary ID are identical. However, as vehicles are killed, the secondary ID for all vehicles possessing a primary ID greater than that of the killed vehicle will be reduced by one. This then allows the vehicles tracks to be re-allocated in a fairly simple manner. This concept can best be summarized by the following variable re-assignment logic.

```
Initially
    $\text{vehicle\_id (i)} = \text{i};$            $\text{i} = 0, \text{N} - 1$
If $\text{k}^{\text{th}}$ vehicle is killed
    $\text{vehicle\_id (i)} = \text{i};$            $\text{i} = 0, \text{k} - 1$
    $\text{vehicle\_id (i)} = \text{i} - 1;$        $\text{i} = \text{k} + 1, \text{n} - 1$
    $\text{vehicle\_id (i)} = \text{dead};$         $\text{i} = \text{k}$
where
    $\text{N} = \#$ of initial vehicles
    $\text{n} = \#$ of remaining vehicles at any given time
```

E. TRACK RE-ALLOCATION

1. Basis of Re-allocation

When the supervisor determines a track re-allocation is necessary, the magnitude of the shift must be calculated and tracks to the new GWPs must be determined and turn time, because any subsequent changes made to the program would have carried through to all experimental runs had they been conducted on the final version

Turn Time (seconds)	% Cleared Targets	% Space Searched
1	99.727 %	98.80 %
2	99.627 %	98.74 %
3	99.364 %	98.82 %
4	99.727 %	98.80 %

Table 3.3 Experimental Results of Varying the Turn Time

As is clearly shown in this data, varying the turn time did not have the significant effect on the simulation as expected. With identical results obtained for turn times of one second and four seconds, it was decided to maintain the turn time setting at one second.

D. VEHICLE IDENTIFICATION PROCEDURES

In order to support the concept of vehicle re-tasking when one is lost, the vehicles utilize a dual identification scheme. Each vehicle is assigned a primary, permanent ID as well as a secondary, changeable ID number, which is based on the number of vehicles remaining in the scenario. The vehicle is tracked and all vehicle data is recorded using its permanent name. For computational purposes and data management internal to the program, the permanent ID of the first vehicle in line is assigned a value of zero and the last vehicle assigned a value of one minus the total number of vehicles. This is done to support the data structure arrays required to execute the code, where the first element is required for it to reach its next assigned GWP. Thinking that this might be too fast, therefore, not accurately reflecting true vehicle motion, a decision to alter the value and determine the effect on the simulation was made. Of particular interest was the impact any increase in this delay time might have on the "shock factor" phenomenon.

Analytical determination of a better value became difficult, as a realistic track would have the vehicle making a sweeping motion around the general vicinity of the GWP. During this type of motion the vehicle travels in both the general direction of the old heading and the general direction of the new heading while making the turn, therefore, determining the exact time lost to the turn becomes somewhat nebulous. Consequently, an experimental approach to the solution, where the delay time was incrementally increased, simulations were run, and the results were examined, was used. Table 3.2 displays the simulation parameters that were used when conducting these experiments.

# of Vehicles	**# of Iterations**	**Vehicle Speed**	**Sensor Width**	**Vehicle Spacing**	**% Overlap**	**Simulation Time**
20	100	5 KTS	4.2672 m (14 ft)	2.5 m (8.202 ft)	70.69 %	7200 sec

Table 3.2 Experimental Simulation Parameters

Table 3.3 presents the data obtained from these experiments. It should be noted that the data gathered during these experiments should be considered example results not final results as the simulations were run on a version of the program other than the final version. However, the data remains viable for determination of the effects of varying the

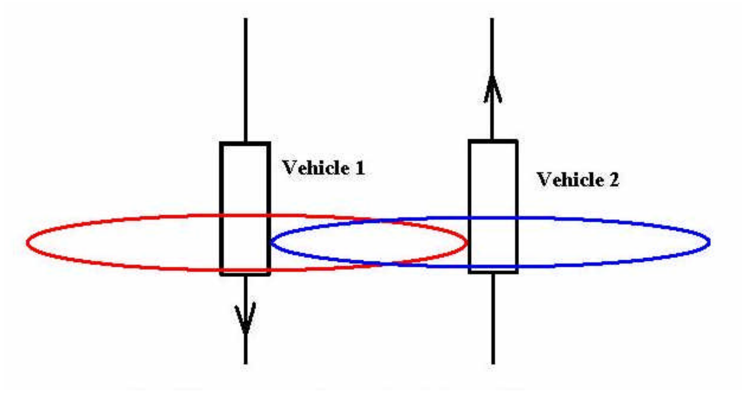


Figure 3.2 Vehicles with 100% Overlap

On the opposite end of the spectrum, Figure 3.3 portrays a pair of vehicles that have zero overlap; half the vehicle spacing equals half the vehicles swath width.

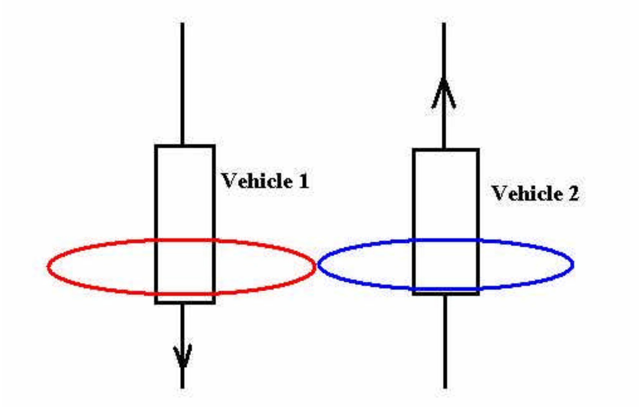


Figure 3.3 Vehicles with 0% Overlap

C. EFFECTS OF VARYING THE TURN TIME

Upon reaching an assigned GWP, the original program allowed for a one second delay to account for the time it took the vehicle to change course to the new heading

However, due to the preliminary nature of this research, all vehicle sensors are assumed to have the same effective range. While sensor width is a user-defined parameter, it is anticipated that typically, the value input will be that of the smallest range whiskers. To do otherwise, would lead to extremely erroneous results being obtained. Furthermore, it is anticipated that follow-on work will account for the different sensor widths, the resulting effects of multiple degrees of overlap, and ultimately the implications it has to the overall simulation results.

2. Calculation of Overlap

As shown previously, overlap is a function of the vehicle spacing and the sensor width. Based on the underlying description of overlap, percent overlap refers to the percentage of the vehicle interval that is swept by multiple vehicle passes. Accordingly, percent overlap is calculated using the following equation:

$$\%\,\text{Overlap} = \frac{\text{Sensor Width - Vehicle Interval}}{\text{Vehicle Interval}} \times 100\,.$$

A zero or negative value, indicating a lack of overlap, will be returned when the vehicle interval is equal to or greater than the sensor width. The more negative the value, the greater the vehicle separation. Also, a value greater than 100% may be achieved if the sensor width is greater than two times the vehicle interval. Figure 3.2 graphically depicts the programs default value of 100% overlap; the vehicle spacing equals half the vehicles swath width.

area. For the purposes of this figure, the same vehicle on different legs of its path generates the overlap. However, it follows that overlap will also result when two separate vehicle's sensors cover the same area.

The horizontal distance between the vertical tracks seen in the diagram represents what is referred to as the vehicle spacing, additionally spoken of as the vehicle interval. As shown in Table 3.1, this spacing is a user-defined parameter in the MVMP simulation, with a default value equal to one half of the sensor width. The input value should be chosen to reflect the desired guidelines for a specific search.

Consequently, it can be seen that percent overlap is a function of both the vehicle interval and the sensor width. Detailed descriptions of sensor width and calculation of percent overlap follow.

1. Sensor Width

Reality holds each sensor having its own limiting range of effectiveness, with the smallest range, in general, being that of the whiskers used to mechanically detect the cables of the moored mines. Sensor widths are a function of the strength of the permanent magnet, the decibel level of the noisemaker, and the length of the whiskers. Additionally, the inter-relationship of these swath widths becomes extremely complicated with multiple degrees of overlap being created and varying resultant effects dependent upon mine type. It is projected that these multiple sensor widths would have a possible impact on the "shock factor" phenomenon and also becomes extremely important when mines with ship counters are considered.

B. PRINCIPLE OF OVERLAP

Chapter II, section C, particularly, Figure 2.3 introduced the principle of overlap and stated its primary uses are for compensation of navigational errors and to help increase the probability of complete clearance. Figure 3.1 below is an expanded view of the section of Figure 2.3 that details this principle.

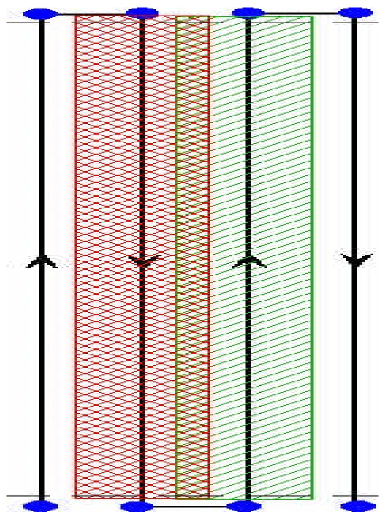


Figure 3.1 Vehicle Overlap

The shaded areas in the figure represent the vehicle's sensor width, also referred to as the swath width, and equates to the range to which a vehicle's sensors are effective. The expression sensor width as used in the simulation is the total horizontal range of the sensors. For example, a vehicle, which has an effective sensor range of five feet from its centerline, would have a ten-foot sensor width. The term overlap refers to that area depicted in the center of the diagram where the sensor widths cover the same geographical

III. PROGRAM EVALUATION

A. PARAMETER SETTINGS

A simulation code, MVMP (Multi Vehicle Minesweep Program), initially developed by J. Kim was used as the basis for this evaluation study. For reference purposes, Table 3.1 lists the critical simulation parameters.

Parameter	Setting	Default Value	Modeling Method
Sensor Width	User Input	-	Input Statement
Vehicle Spacing	User Input	½ Sensor Width	Input Statement
Turn Time	1 sec	-	Fixed
Navigational Error	0 – 2 m	-	Random Number
Mine Placement Position Error	0 – 1 m	-	Random Number
Influence Mine “Shock Factor”	25 m radius	-	Fixed
Moored Mine “Shock Factor”	10 m radius	-	Fixed
Delay Time to Destroy Moored Mines	15 Sec	-	Fixed
Vehicle Delay Time	Function[1]	-	Calculated
Vehicle Speed	User Input	-	Input Statement
Overlap	Function[2]	100%	Calculated
Number of Vehicles	User Input	-	Input Argument
Number of Iterations	User Input	-	Input Argument
Length of Simulation	User Input	21600 sec	Input Statement

Table 3.1 Program Parameter Settings

[1] Function of influence mine “shock factor”, vehicle speed and the delay time to destroy the moored mines, calculated using the equation:

$$\text{delay} = \frac{\text{influence mine "shock factor"}}{\text{vehicle speed}} + \text{destruction delay time} + 1 \ .$$

[2] Function of vehicle spacing and the vehicle sensor width. Further detail on this parameter follows later in the chapter.

THIS PAGE INTENTIONALLY LEFT BLANK

demonstrate the distinction between the mine types used in the scenario. In reality the mine would determine this for the vehicle by detonating and destroying the vehicle if it were an influence mine. If the sensed mine is a moored mine, the report reflecting this is shown going out to the supervisor. The diagram also illustrates the vehicle's decision to change its track once it receives an update message from the supervisor. The vehicle continually repeats the process until it is killed, its battery runs out, or the search time is expired.

The second and third logic diagrams are contained within each vehicle block, and represent diagrams specific to that vehicles operations. Dashed lines represent all internal communications and the solid lines represent all logic flow. In a broader view of the scenario, the third diagram would be repeated for as many swimmer vehicles as were utilized for that operation, and the supervisor vehicle would receive input through its communications port from all of the swimmer vehicles, as reflected in figure 2.1.

The second diagram, that of the supervisor vehicle, shows the vehicle state processor and its link to the decision process within the supervisor. As communication is lost with a vehicle, the supervisor determines which vehicle was killed, determines if any vehicles are remaining, if so, re-assigns tracks based on GWPs. The process is repeated until the search time runs out or there are no more vehicles remaining. Loss of communications with a vehicle is initiated either by the loss of a signal from a vehicle or by a report of a vehicle locating and starting the demolition process of a moored mine. Additionally, it is envisioned that the vehicle state processor could as necessary initiate a query process of all vehicles to determine their status.

The third diagram shows the swimmer vehicle logic assuming it is following its assigned track and sending a signal to the supervisor showing that it remains alive until either sensing a mine or receiving an updated track. For the purposes of the flow chart, the term sensed is used to distinguish between the vehicle detecting a moored mine or an influence mine detecting the swimmer. Once the vehicle senses a mine, the chart shows a logic step of determining whether the mine is an influence mine or not. This is a step to

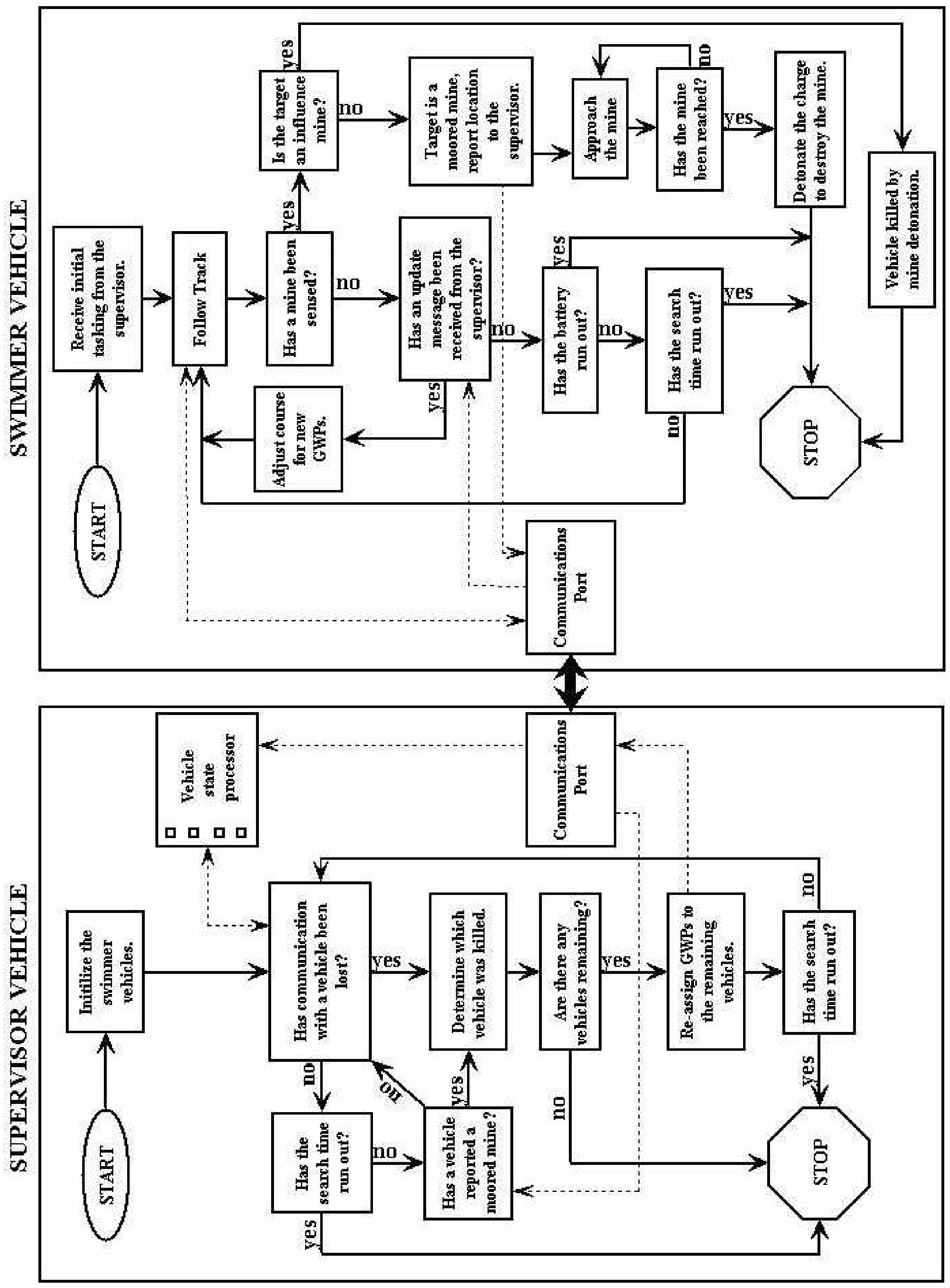

SWIMMER VEHICLE
START
Receive initial tasking from the supervisor.
Follow Track
Has a mine been sensed?
yes
no
Is the target an influence mine?
yes
no
Target is a moored mine, report location to the supervisor.
Approach the mine
Has the mine been reached?
no
yes
Detonate the charge to destroy the mine.
Vehicle killed by mine detonation.
Has an update message been received from the supervisor?
yes
no
Adjust course for new GWPs.
Has the battery run out?
yes
no
Has the search time run out?
yes
no
STOP
Communications Port
SUPERVISOR VEHICLE
START
Initilize the swimmer vehicles.
Has communication with a vehicle been lost?
yes
no
Vehicle state processor
Determine which vehicle was killed.
Are there any vehicles remaining?
yes
no
Re-assign GWPs to the remaining vehicles.
Has the search time run out?
yes
no
Has the search time run out?
yes
no
Has a vehicle reported a moored mine?
yes
no
STOP
Communications Port

Figure 2.5 Logic Flow Chart for Robotic Minesweeping

proceeding along its track, thinks the swimmer is a ship, and detonates itself, destroying the swimmer in the process.

Further requirements for these vehicles would be small data processors linked with underwater navigational equipment in order for the vehicles to maintain track. In addition to monitoring and controlling its own functions, the data processor requirements are to receive information from the supervisor, determine the appropriate response, and rapidly execute any required actions. The processor must also be able to store a minimal amount of information, in particular the assigned track and associated GWPs. Precise navigational requirements for these vehicles have yet to be determined, however, it is recognized that to preserve the integrity of the minesweeping operation, fairly accurate systems would be required. For the purposes of this research, a navigational error up to two meters was assumed. This is modeled as a random position offset from the desired track.

E. LOGIC

Fundamental to understanding the complex nature on which this concept is built is to comprehend the logic of its basis. Figure 2.5 displays this logic in an easy to follow graphical format (see page 15). This logic diagram is best considered as three diagrams built into one.

The first of these three diagrams being a simple two block diagram linking the swimmer with the supervisor. Each of the vehicles is represented by one of the two large outer boxes of the diagram. The large double-sided arrow connecting the communications port of each represents the two-way communications link between the vehicles.

While not accounted for in this scenario, and as previously discussed, this multiple pass behavior increases the probability of complete minefield coverage as well as helping account for mines that are equipped with such devices as ship counters.

2. Swimmer Vehicles

As previously mentioned, the primary design consideration for the swimmers is their expendability, so minimization of onboard equipment, sensors, and processors becomes a priority.

In order to detect and sweep the mechanically activated tether mines the vehicles will carry mechanical probes (whiskers) that extend from the vehicle in the horizontal plane. The whiskers will be long (greater than five feet is envisioned); slender rods having a hooked shaped end. The tethered mines are detected when the swimmers proceed along track and the mechanical probes strike and hook the mooring cable of one of the moored mines. The swimmer subsequently reports to the supervisor the detection of a moored mine. The vehicles propulsion control system then causes the swimmer to steadily climb the mooring cable to reach the mine casing. Once the vehicle senses that it reached the mine casing, detonation of an onboard explosive destroys the mine and the swimmer.

For sweeping of the bottom lying influence mines, the vehicle must be capable of generating both magnetic and acoustic signatures, which adequately represent that of a ship. To do this, the vehicles carry a sufficiently strong permanent magnet apparatus and noise-making device. The sweeping of the mine occurs when it detects the swimmer allocations more pronounced. Tracks that end in the field represent loss of a vehicle to a mine.

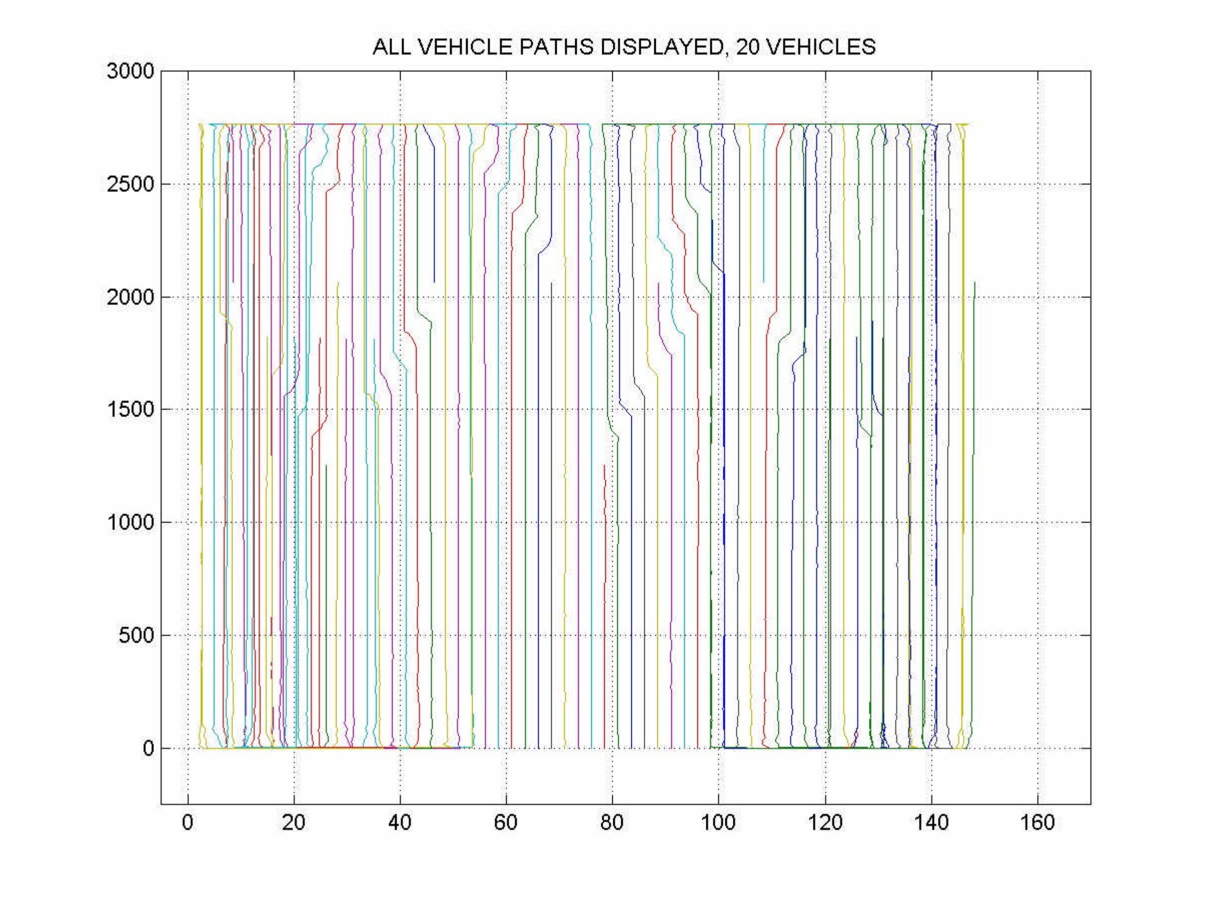


Figure 2.4 Swimmer Vehicle Tracks for a 20 Vehicle Simulation

Also particularly noticeable in this figure, especially on the left hand side, the supervisor recognizing that a swimmer reaches the constraint imposed by the boundary of the minefield, assigns a track such that the vehicle will re-sweep areas previously covered. This action continues as long as battery power and operational time constraints allow.

vehicle relays progress reports to and maintains communications with a control ship through an acoustic modem and/or a line of sight radio link.

For the purposes of the research conducted in support of this thesis, the initial tracks from the launching platform to the rendezvous point are not accounted for. Additionally, no consideration is given to the limitations of communications between the supervisor vehicle and a control platform. Since this research is considered preliminary, it was decided that these factors should be disregarded for now, and accounted for in follow-on research..

1. Supervisor Vehicle

The larger, and more complex supervisor vehicle's primary design criterion revolves around a vehicle state processor, utilized to analyze the data received from the swimmer vehicles. The onboard computers carry all pertinent data regarding the minefield, including the GWPs used to assign the swimmer vehicle tracks. The vehicle state processor monitors all transmissions from and queries the swimmer vehicles as necessary in order to determine if any vehicle was killed due to a mine explosion. If a swimmer was destroyed, the processor must determine which vehicle it was and then assign new tracks to all remaining vehicles. Additionally, the supervisor must monitor the search time, and vehicle progress, and re-call any remaining vehicles when the search reaches the end of the specified parameters. As an example, figure 2.4 shows a two-hour, twenty-vehicle simulation and the dynamical re-allocation of tracks to the swimmer vehicles by the supervisor. It should be noted that the axes are not set equal which makes the track re-

Positioning System (GPS) points, also known as Global Way Points of planned tracks (GWPs) as assigned by the supervisor vehicle. The heavy black lines connecting the GWPs depict the ideal track for a swimmer. Additionally, the shaded areas illustrate the effect of sensor width overlap along the vehicle tracks. The subject of overlapping is more thoroughly addressed later in the text.

D. VEHICLE DESCRIPTIONS

The AUVs and/or UUVs conceptualized are to be small enough and portable enough that the fleets of these minesweeping vehicles may be transported and launched from virtually any US Navy or Naval Force ship. Additionally, the swimmer vehicles are considered to be expendable in that they may be destroyed in the process of minesweeping.

Once a potential minefield is identified for reconnaissance and/or neutralization, the fleets are to be pre-programmed to swim to a specified GWP, and then launched from either a single ship or multiple ships. Upon reaching the assigned GWP the swimmer vehicles are to be coordinated and controlled by the supervisor vehicle for the mine countermeasures operation.

Communications between the supervisor vehicle and the swimmer vehicles exist primarily through acoustic modem transmissions. If the vehicles were to be operating in extremely shallow water, such as immediately in front of a beachhead that will be utilized for an amphibious landing, and a radio antenna were able to broach the ocean surface, then line of sight radio communications also may be utilized. Additionally, the supervisor the probability of complete clearance ($P_{overall}$) is a function of the vehicle's probability of detection (p). Additionally, multiple passes over an area significantly increase the probability of a complete sweep of the minefield. Assuming m passes are made over the same area, the probability of complete clearance becomes

$$P_{overall} = [1-(1-p)^{m}].$$

With this in mind, and considering the use of overlap to help account for navigational errors and to help increase the number of passes over an area, a lawnmower complete area search pattern with overlap was chosen. Figure 2.3 demonstrates the basic concept of the lawnmower search pattern as utilized.

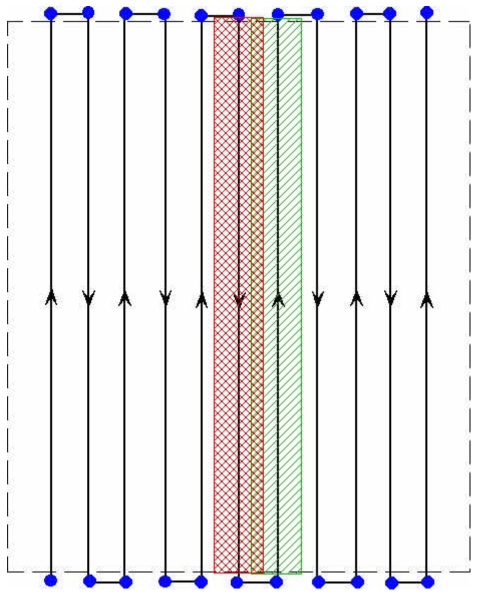
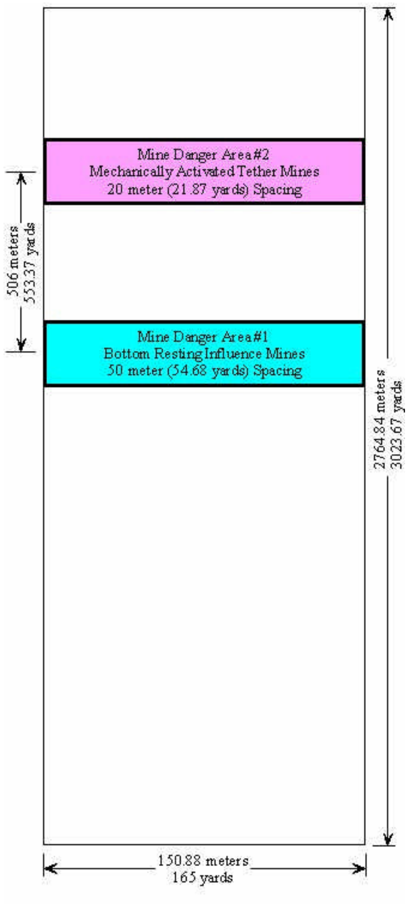


Figure 2.3 Lawnmower Search Pattern

The dashed line in the figure represents the boundaries of the portion of the minefield being shown. The dots just outside of the minefield boundaries are the Global



Mine Danger Area#2
Mechanically Activated Tether Mines
20 meter (21.87 yards) Spacing
Mine Danger Area#1
Bottom Resting Influence Mines
50 meter (54.68 yards) Spacing
506 meters
553.37 yards
2764.84 meters
3023.67 yards
150.88 meters
165 yards

Figure 2.2 Minefield Layout

When a simulation is conducted, the computer code randomly generates a minefield that conforms to these constraints.

C. SEARCH PATTERN

Koopman (1980) and Stone (1975) discuss various methods of search revealing their extremely complex natures, and show that each has its own advantages and disadvantages. As discussed in Healey (1999), a linear relationship between percent coverage and search time is obtained when using a complete area search method. As well, Should additional vehicles be within this footprint during detonation, multiple vehicles would be lost to a single mine. Obviously, this phenomenon is undesirable and consequently the staggered start times were developed.

Detailed descriptions of the various aspects of the scenario utilized for this research, as provided by Mr. Lee Fry of Coastal Systems Station, Panama City FL, is discussed in the following sections of this chapter.

B. MINEFIELD

In a X-Y coordinate system with the X coordinates being viewed in the horizontal direction and the Y coordinates being viewed in the vertical direction, the minefield for the simulation is sized 150.88 meters (165 yards) by and 2764.84 meters (3023.67 yards). The minefield contains two parallel mine lines spaced 506 meters (533.37 yards) apart. The first mine danger area is assumed to contain bottom resting influence mines that are triggered by either the magnetic or acoustic signature of a passing vessel. The mines within this zone area are spaced 50 meters (54.68 yards) apart. Mine danger area two contains mechanically activated, tethered mines that are triggered when a vessel strikes a mechanical probe protruding from the mine casing. Mines in this line are spaced 20 meters apart. Figure 2.2 graphically depicts this scenario.

shifts its course to reflect the change. A typical vehicle layout for a twenty-vehicle fleet is shown in Figure 2.1.

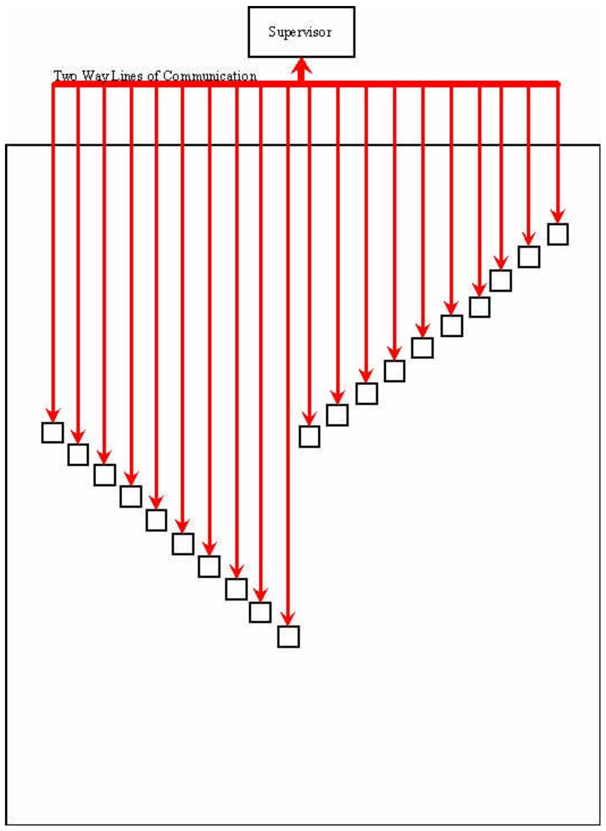


Figure 2.1 Typical Layout for a 20 Vehicle Scenario

The staggered formation of the vehicles results from incorporating delay times between the vehicles starting their search. Principally this is done to allow the vehicles to spread out enough to minimize the effects of the "shock factor" phenomenon. "Shock factor" is best characterized as the range of influence of a particular mine when it detonates and is a function of the charge strength of the particular mine in question.

II. CONCEPT

A. OVERVIEW

In an effort to develop a viable mine warfare countermeasure technique that can be rapidly and effectively deployed in support of the contingency operations as previously mentioned, the use of a multi-vehicle fleet of autonomous underwater vehicles linked to a supervisor vehicle through two-way paths of communication is being considered. Within this theory, the supervisor vehicle is to be stationed outside of the designated minefield yet maintain communications links with all the swimmer vehicles. While not specifically addressed, and not really relevant to this study, it is noted that the supervisor could be either stationary or mobile depending upon the tactical situation of the specific clearance operation being conducted. Additionally, use of multiple relay stations could be incorporated into the scenario if necessary to increase the range of the communications links beyond what is technologically feasible for single point links. The swimmer vehicles are to be inexpensive, expendable, and carry a minimum amount of equipment necessary to conduct mine clearance operations. These vehicles through various methods will sense and detonate a mine, thereby neutralizing the mine, and would in the process be lost. When a vehicle is lost to the neutralization of a mine or otherwise looses communication with the supervisor vehicle (such as through the loss of battery power) the supervisor vehicle then re-allocates assigned tracks to the remaining vehicles in order to ensure maximum coverage of the minefield. Once a swimmer vehicle receives updated tasking, it

Chapter IV shows results that prove that the program is effective. Simulations were conducted using various different combinations of simulation time and number of vehicles. Additionally, some results are displayed graphically and the results are discussed.

Chapter V summarizes the conclusions of this research as laid out in the previous chapters. Additionally, recommendations for further study are made.

Subsequently, there are three appendixes attached to this report. Appendix A is the coded program file. Appendix B is a small Matlab file used for generating graphs of the output data from the program. Appendix C is a users guide to the program.

B. SCOPE OF THIS WORK

The overall problem of robotic minesweeping is complex and diverse involving many different scenarios, mine types and various other contributing factors. This study will focus on the evaluation and implementation of the computer simulation, Multi Vehicle Minesweep Program (MVMP), for mine clearance operations using multiple swimming robotic vehicles. Their actions are coordinated through a supervisor vehicle in order to ensure maximum minefield coverage. In particular, as mines are cleared and thus a vehicle killed, the remaining vehicles will be re-tasked by the supervisor vehicle to close the gap left open by the explosion. The purpose of this thesis is threefold:

1. To analyze a previously coded computer simulation.
2. To implement any necessary changes in the computer simulation.
3. To run simulations and evaluate the output.

Chapter II discusses the concept of robotic minesweeping and provides a detailed description of the concept and aspects of the scenario. Included discussions concentrate on the minefield layout and mine types, the search pattern utilized, the vehicles and their operations, and the control logic for the simulation.

Chapter III describes various fundamental aspects of the program. In particular, it addresses the critical issues of vehicle overlap, assignment of vehicle identification numbers, track re-assignment, including calculations of the distance required for vehicle shift, and the effects of varying the vehicle's turn time. Additionally, a table showing the critical simulation parameters and their respective modeling methods is included.

the coming years compel us to develop a Naval Mine Warfare capability that will ensure the safe operation of our forces in hostile littoral regions." (Crute, 2000) Current mine countermeasure methods involve the use of highly specialized forces using expensive equipment and ships. This often results in personnel having to enter the minefield in order to neutralize a mine. Additionally, such methods tend to be extremely time consuming and cannot be guaranteed to have neutralized all potential threats to Naval forces. Ultimately, it is desired that any force could rapidly and as effectively as possible, neutralize a minefield with a minimum of risk involved, thereby, allowing for the rapid deployment of forces in any contingency operation. One solution is "for operations that require the clearance of mines over large areas, a cooperative engagement approach, utilizing remote and autonomous vehicles from multiple platforms will facilitate the rapid and thorough sanitization of the area." (Crute, 2000) Additionally, it is noted "Advances in robotics, artificial intelligence, and underwater communication will lead to greatly reduced risk to personnel in the dangerous business of mine countermeasures. We will progressively remove the man from the minefield. Remote or autonomous vehicles will map minefields and, if necessary, neutralize mines. High value assets will transit cleared areas with minimal risk from enemy mines." (Crute, 2000) The research conducted in support of this thesis is the beginning of one such approach to mine countermeasure operations.

I. INTRODUCTION

A. BACKGROUND

Due to rapid increases in technological developments over the last several years, small Autonomous Underwater Vehicles (AUVs) and Unmanned Underwater Vehicles (UUVs) are attractive for use in scientific, commercial and military applications in the ocean. One of the greatest benefits of these vehicles is the removal of the tether that has previously been required for the transfer of sensor data and control of underwater robotic vehicles. This removal allows the vehicles to travel into areas that were previously considered unreachable due to depth or extent, where the tether may simply have become fouled, or inherent danger existed which could have damaged either the tether or the controlling ship. As such, since the launching ship can remain at a reasonably safe distance, the attractiveness of these vehicles for use in minesweeping operations becomes readily apparent.

With the changing global structure and the nature of potential adversaries, US Naval Forces continue to place a greater emphasis on littoral warfare, which necessitates the increasing need for effective, relatively inexpensive, and rapidly deployable mine countermeasure operations. As stated in NAVAL MINE WARFARE VISION 2010, "Since most mines are inexpensive and easy to obtain, many potential adversaries possess a significant mining capability. The complexities of warfare in the littoral regions, the need for rapid response to crises throughout the world, and the reduction of our force levels in

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGMENTS

I would like to acknowledge Mr. Lee Fry and Mr. Jeff Rish of the Coastal System Station, Panama City, FL for providing the financial support and guidelines for this research. I would also like to thank the staff members, faculty, and fellow students who supported me during my time here at the Naval Postgraduate School; especially, Professor Tony Healey for his untiring guidance, constant support, and patience during the thesis process. Additionally, I would like to acknowledge Joung K. Kim who initially coded the computer program on which this research was based.

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

TABLE OF CONTENTS